CS647 Modern Storage Systems

- LSM Analysis and Tradeoffs
- Original LSM
- VAT analysis
- How much memory for index?
- What is the optimal system cost? (5-minute rule)

Angelos Bilas, bilas@ics.forth.gr

CS647 Modern Storage Systems, Fall 2022
(Modern Key-Value Store Design and Implementation)
Angelos Bilas, bilas@csd.uoc.gr
Computer Science Department, University of Crete

LSM: Optimal growth factor and #levels

[VAT: Asymptotic Cost Analysis for Multi-Level Key-Value Stores. Nikos Batsaras et al. February 2020. arXIV 2003.00103. arxiv.org/abs/2003.00103v1]

- LSM amplification is sum of
  - Level amplification (better for larger f → fewer levels)
  - Merge amplification (better for smaller f → more amortization)
- What is the optimal growth size f?
- Definitions
  - Ki, Vi: Size of all keys, values
  - Si: capacity of level I (Ki+Vi)
  - l: number of levels on storage device
  - f: growth factor
  - H: Required system throughput in pages/s
  - R: Application throughput (sustained) in bytes/s
  - Sp: page size (bytes/page) for device to achieve R
- LSM: contribution of each level to compaction is
  - Each level is read and written twice
  - Except last level

$$H = \frac{R}{S_p}((2f_1 + 2) + \ldots + (2f_l + 1)) \qquad (18)$$

$$H = \frac{R}{S_p}(2(f_1 + f_2 + \ldots + f_l) + 2l - 1)$$

$$= \frac{R}{S_p}(2\sum_{i=1}^{l} f_i + 2l - 1)$$

- Optimal case is when H is minimum

$$\min H = \min(2\sum_{i=1}^{l} f_i + 2l - 1) \qquad (20)$$

- Trivially, will lead to S0=Sl, l=0, or Sl=0
  - Need to minimize H under some constraint

- (a) When S_l/S_0 is constant (memory cost)
  - Find optimal f for storing data (S_l) fixed times more than memory

$$\min_{\frac{S_l}{S_0}=C} H = \min_{\frac{S_l}{S_0}=C}(2\sum_{i=1}^{l} f_i + 2l - 1) \qquad (21)$$

  - Result: optimal f=~4.48 (~4-5)
    - 1 16 256 (f=16)
    - 1 4 16 64 256 (f=4) **<- best**
    - 1 2 4 8 16 32 64 128 256 (f=2)
    - 2.56 25.6 256 (f=10) <- ???
      - (not considered by optimization problem)
- (b) When $S_0+S_1+\ldots+S_l$ is constant (store capacity)
  - Find optimal f for fixed device budget ($S_0+\ldots+S_{l-1}$) – dataset size S_l can vary
  - Alternatives may have fewer or more levels but all need to have the same device space

$$\min_{S_0+\ldots+S_l=C} H = \min_{S_0+\ldots+S_l=C}(2\sum_{i=1}^{l} f_i + 2l - 1) \qquad (22)$$

  - Result: optimal f=~10-11
    - 1 120 (S=~121, f=~100)
    - 11 110 (S=~121, f=~10) **best**
    - 1 3 9 27 81 (S=~121, f=~3)
    - 3 18 100 (S=~121, f differs)
    - 0.355 1.4 6 23 91 (S=~121, f=~4) ???
      - (solution says this is suboptimal)
  - This is what RocksDB uses by default
- LSM paper discusses both (a) and (b)
  - VAT paper includes all step of the derivations

LSM: Optimal growth factor and #levels

Constant memory $C = S_l/S_0$

- => $C = f^l$ => $l = \log_f C$ and $f = l\text{-root}(C)$

$$\min_{\frac{S_l}{S_0}=C} H = \min_{\frac{S_l}{S_0}=C}(2lf + 2l - 1) = \min_{\frac{S_l}{S_0}=C}(2l \cdot \sqrt[l]{C} + 2l - 1)$$

- To minimize solve for derivative = 0

$$\frac{d}{dl}(2l \cdot \sqrt[l]{C} + 2l - 1) = 0 \Rightarrow 2\sqrt[l]{C} - \frac{2\sqrt[l]{C} \cdot \log_e C}{l} + 2 = 0$$

$$\Rightarrow \sqrt[l]{C}\left(\frac{\log_e C}{l} - 1\right) = 1 \Rightarrow (e^{\log_e C})^{\frac{1}{l}}\left(\frac{\log_e C}{l} - 1\right) = 1$$

$$\Rightarrow e^{\frac{\log_e C}{l}}\left(\frac{\log_e C}{l} - 1\right) = 1 \xrightarrow{\frac{\log_e C}{l} = x} e^x(x-1) = 1$$

$$\Rightarrow (x-1)e^x e^{-1} = e^{-1} \Rightarrow (x-1)e^{x-1} = \frac{1}{e}$$

- With Lamberts function, by definition $W(xe^x)=x$

$$W((x-1)e^{x-1}) = W(\frac{1}{e}) \Rightarrow x - 1 = W(\frac{1}{e}) \xrightarrow{x = \frac{\log_e C}{l}} \Rightarrow$$

$$\frac{\log_e C}{l} - 1 = W(\frac{1}{e}) \Rightarrow l = \frac{\log_e C}{W(\frac{1}{e}) + 1}$$

- Therefore, optimal f is

$$f = \sqrt[l]{C} = C^{\frac{1}{l}} = C^{\frac{1}{\frac{\log_e C}{W(\frac{1}{e})+1}}} = C^{\frac{W(\frac{1}{e})+1}{\log_e C}}$$

$$= (C^{\frac{1}{\log_e C}})^{W(\frac{1}{e})+1} = (e^{\log_e C \frac{1}{\log_e C}})^{W(\frac{1}{e})+1}$$

$$= e^{W(\frac{1}{e})+1} \qquad (24)$$

- Given that W(1/e) is about 0.5, we can get optimal **f = $e^{3/2}$ = ~4.48**
  - Simpler solution

$$\min_{\prod_{i=1}^{l} f_i = C}(2lf + 2l - 1) \approx \min_{\prod_{i=1}^{l} f_i = C}(lf)$$

  - Optimal $l = \log_e C$ => $e^l = C$. Since $f^l = C$ => $f = e$ ~= 2.718

$$\frac{d}{dl}\left(l \cdot \sqrt[l]{C}\right) = \sqrt[l]{C} + l\left(\log_e C \cdot \sqrt[l]{C} \cdot (-\frac{1}{l^2})\right)$$

$$= \sqrt[l]{C}\left(1 - \frac{\log_e C}{l}\right) = 0 \Rightarrow l = \log_e C \qquad (25)$$

Constant total capacity $C = S_0 + \ldots + S_l$

$$\min_{S_0 + \ldots + S_l = C} H = \min_{S_0 + \ldots + S_l = C}(2\sum_{i=1}^{l} f_i + 2l - 1) \approx \min_{\sum_{i=0}^{l} S_i = C} \sum_{i=1}^{l} f_i$$

- Minimize with Lagrange multipliers
  - Masters thesis of Nikos Batsaras and arxiv TR

$$f_{l-2} = f_{l-1} + \frac{1}{f_{l-1}}$$

Similarly:

$$f_{l-3} = f_{l-2} + \frac{1}{f_{l-1}} \cdot \frac{1}{f_{l-2}}$$

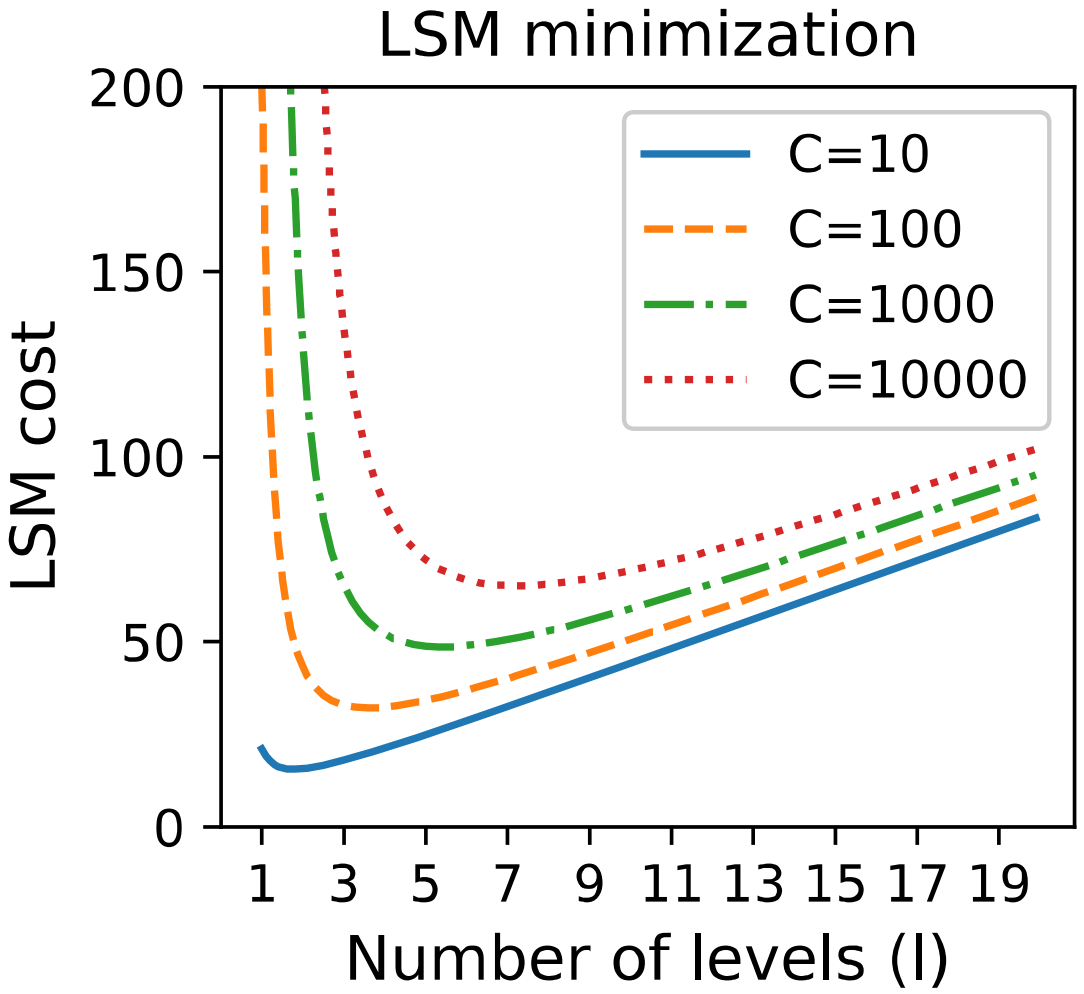
$$\ldots$$

$$f_1 = f_2 + \frac{1}{f_{l-1}} \cdots \frac{1}{f_2}$$

- E.g. l=5
  - f1=11.0998, f2=11.0991, f3=11.0909, f4=11, f5=10
  - For all practical purposes all equal to 10

LSM: Optimal growth factor and #levels

- Amplification vs. number of levels for various values of $C=S_l/S_0$



(a) LSM

VAT Analysis

[VAT: Asymptotic Cost Analysis for Multi-Level Key-Value Stores. Nikos Batsaras et al. February 2020. arXIV 2003.00103. arxiv.org/abs/2003.00103v1]

- High R is achieved with large and sequential Sp
- Say *some compaction technique* uses
  - (1) smaller I/Os and
  - (2) more random I/Os
  - Then, it can only achieve a percentage r of R
  - r=1 is max throughput and "pure" LSM analysis
- Hopefully, this compaction technique reduces also compaction traffic to a percentage **a** of full traffic due to key skew
  - Otherwise, there is no point in using smaller and random I/Os
  - a=1 is full traffic and "pure" LSM
- Then, we can calculate traffic D and time T for compaction as follows

Backup: Modeling I/O Traffic for Leveling Compaction

Definitions:

K_i , V_i : Size of keys, values of level i.

S_i : Capacity of level i $(K_i + V_i)$.

$\grave{}$: Number of device resident levels.

We calculate the I/O traffic as follows:

$$D = \frac{S_{\grave{}}}{S_0}(S_0) + 2\sum_{j=1}^{S_{\grave{}}/S_0}((j-1) \bmod f) \cdot S_0$$

$$+ \frac{S_{\grave{}}}{S_1}(2S_1) + 2\sum_{j=1}^{S_{\grave{}}/S_1}((j-1) \bmod f) \cdot S_1$$

$$+ \ldots$$

$$+ \frac{S_{\grave{}}}{S_{\grave{}-1}}(2S_{\grave{}-1}) + 2\sum_{j=1}^{S_{\grave{}}/S_{\grave{}-1}}((j-1) \bmod f) \cdot S_{\grave{}-1}$$

Backup: Minimization Problem Derivation

We plug the merge amplification parameter a in I/O traffic equation:

$$D = \frac{S_\ell}{S_0}(S_0) + 2a\sum_{j=1}^{S_\ell/S_0}((j-1) \bmod f)\cdot S_0$$

$$+ \ldots$$

$$+ \frac{S_\ell}{S_{\ell-1}}(2S_{\ell-1}) + 2a\sum_{j=1}^{S_\ell/S_{\ell-1}}((j-1) \bmod f)\cdot S_{\ell-1}$$

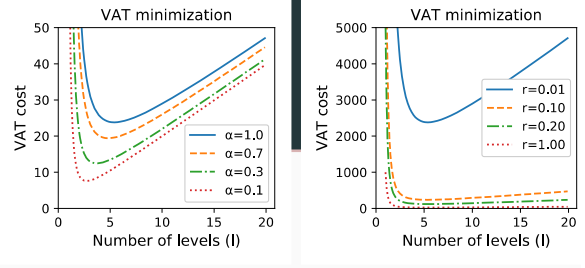
We calculate:

- Time to write the data of size S_l as: $T = \frac{D}{r \cdot R_{opt}}$.
- Optimal time $T_{opt} = \frac{S_\ell}{R_{opt}}$.
- The ratio: $\frac{T}{T_{opt}} = \frac{2^\ell - 1 - a^\ell + af^\ell}{r}$.

The goal:

- Minimize $\frac{T}{T_{opt}}$ i.e. get as close to optimal execution time as possible.

VAT: Analysis for Leveling Compaction

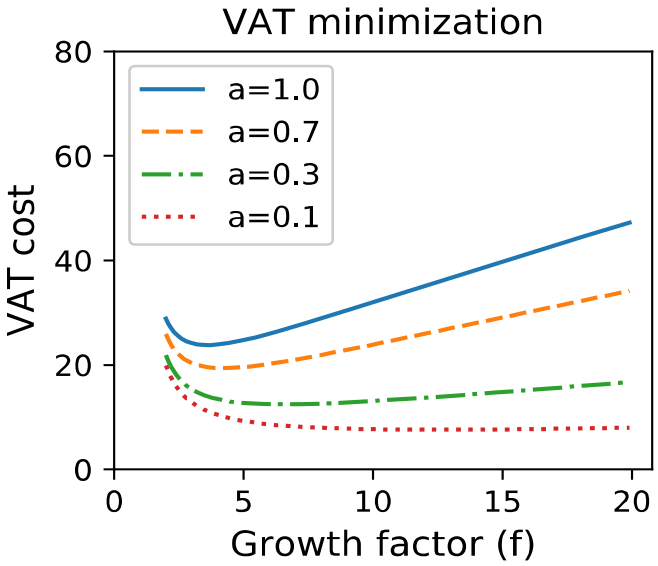


(a) r = 1, C = 1000 (b) a = 1, C = 1000

Figure 2: (a) VAT cost ratio (T/T_{opt}) for optimal throughput ($r = 1$) and different values of merge amplification a. (b) VAT cost ratio at maximum merge amplification ($a = 1$) and different values of achieved device throughput r.

The VAT analysis:

- Defines I/O cost.
- Results in a cost minimization problem: $\frac{2^{l} - 1 - a^{l} + af^{l}}{r}$.



(a) $r = 1$

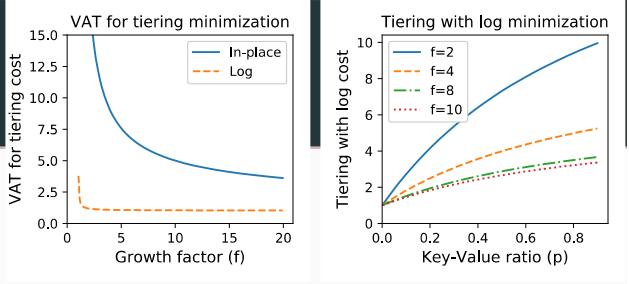
Observations:

- Curves have a minimum point.
- Optimal growth factor decreases as merge amplification increases.

VAT: Analysis for Tiering Compaction

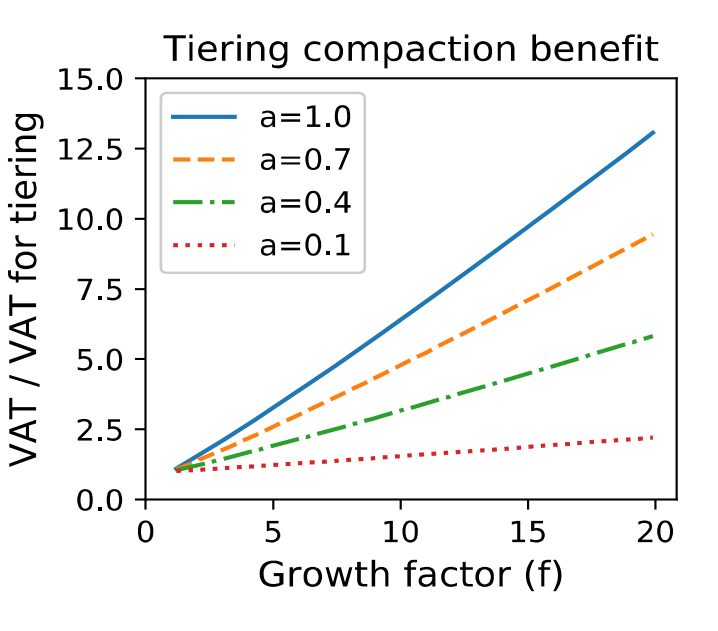
- Cost minimization problem: $\frac{2^{`}-1}{r}$.
- KV separation benefit:

$$\frac{\textit{VAT for leveling}}{\textit{VAT for tiering}} = \frac{\frac{2^{`}-1-a^{`}+af^{`}}{r}}{\frac{2^{`}-1}{r}}$$



(a) p = 1%, r = 1, C = 1000 (b) r = 1 C = 1000

Figure 5: (a) VAT for tiering. (b) VAT for tiering with values in log.



(b) $r = 1$

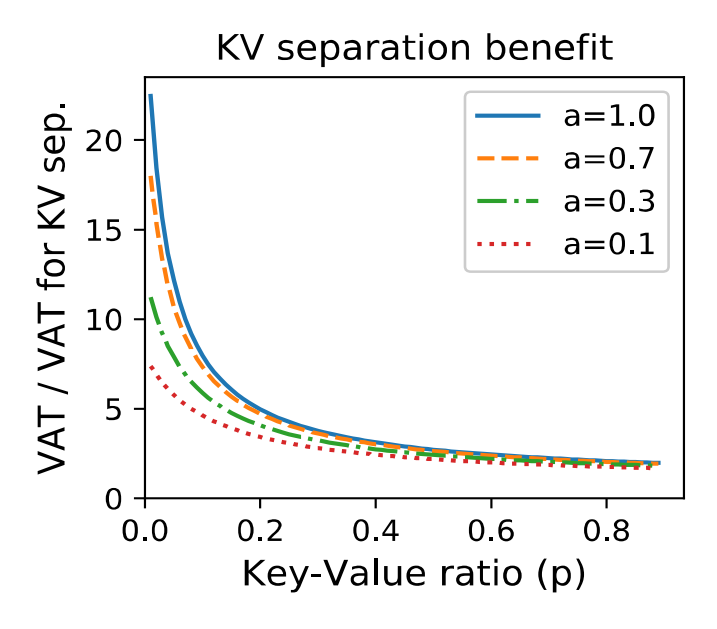
Observations:

- Less amplification than leveling compaction ($\nearrow$ 5x).
- Benefit deteriorates as *a* decreases.

VAT: Analysis for KV Separation

- Cost minimization problem: $\frac{p(2^{\lceil} - 1 - a^{\lceil} + af^{\lceil}) + p + 1}{r \cdot (p+1)}$, $p = \frac{Key\ size}{Value\ size}$.
- KV separation benefit:

$$\frac{VAT\ for\ values\ in\text{-}place}{VAT\ for\ KV\ separation} = \frac{\frac{2^{\lceil} - 1 - a^{\lceil} + af^{\lceil}}{r}}{\frac{p(2^{\lceil} - 1 - a^{\lceil} + af^{\lceil}) + p + 1}{r \cdot (p+1)}}$$

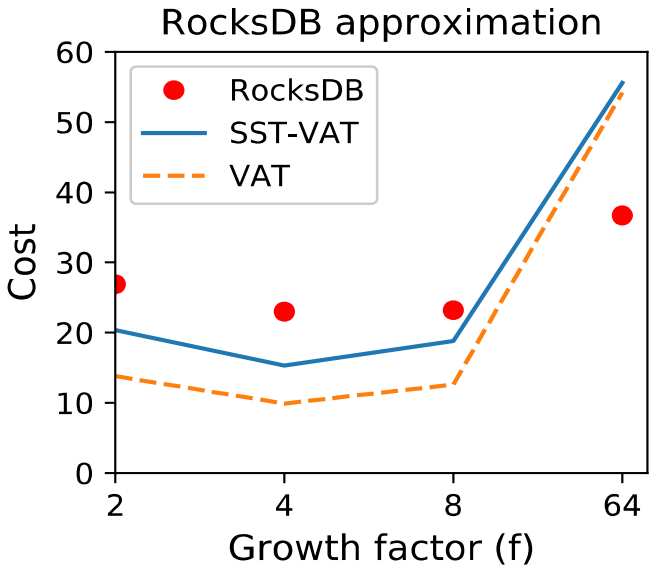


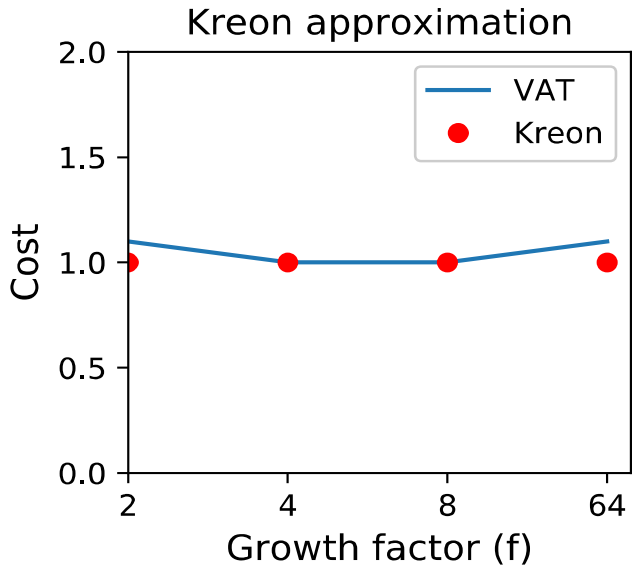
(c) $f = 8$

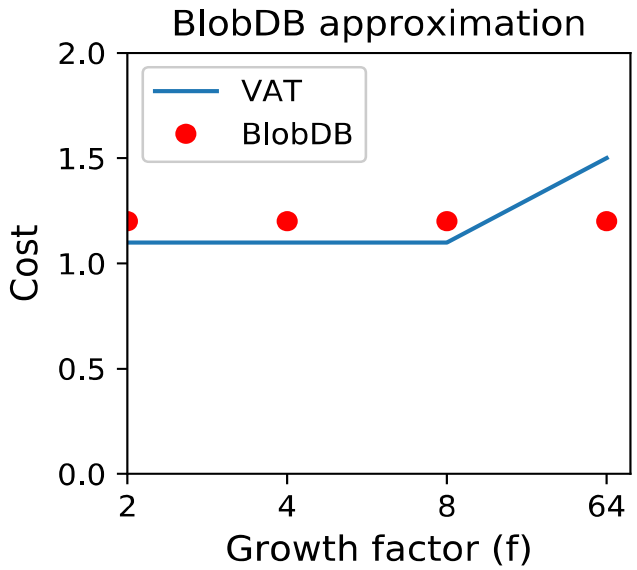
Observations:

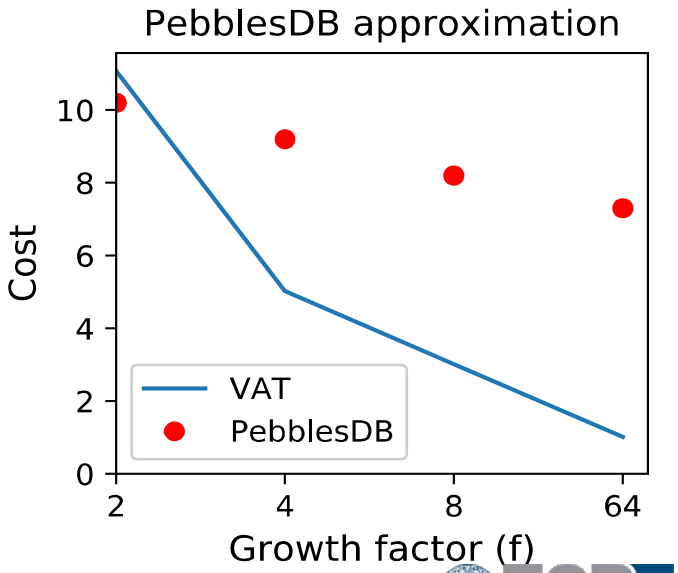
- Less amplification ($\uparrow$ 10x).
- Benefit increases as p decreases.
- Benefit decreases with a.

Experimental Results









nikbats@cs.forth.gr

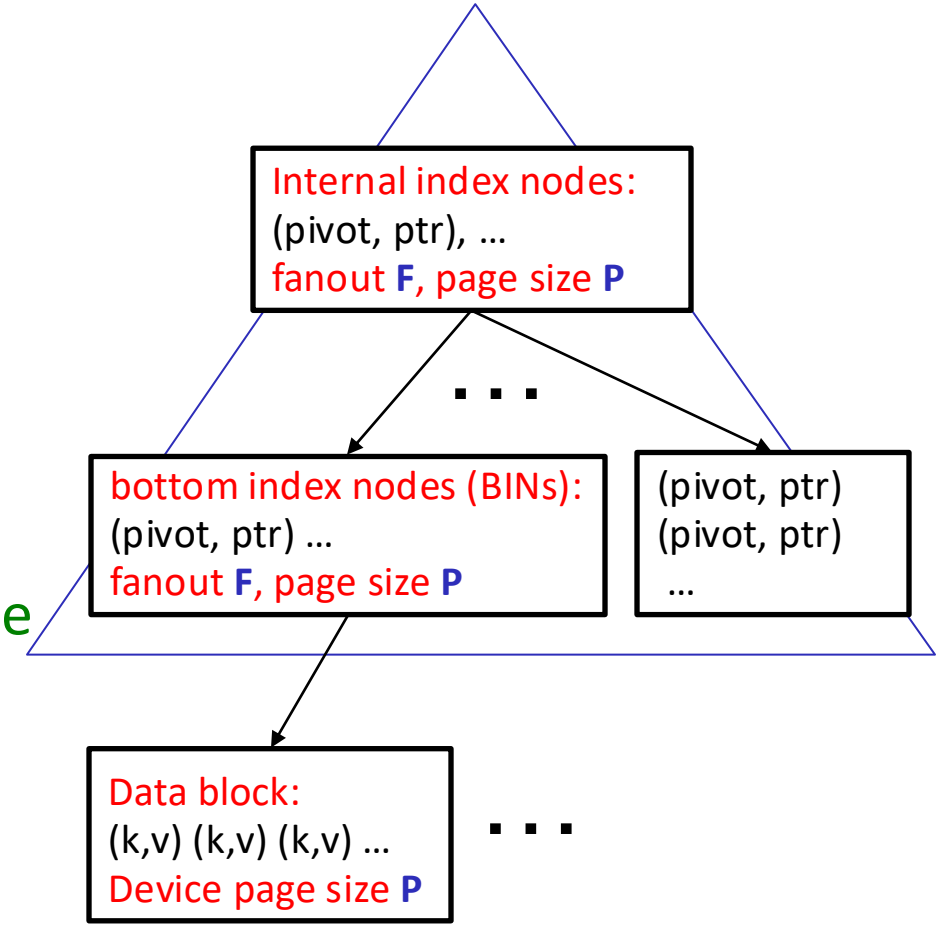
Optimal way to reorganize data?

- So far we assume KV store has levels with growing size by factor f
  - L_0 fits in memory (compulsory)
  - In some sense, we limit the LSM structure and reorganization possibilities
- What if we do not assume anything about levels and their size and how merge occurs?
  - What is the optimal way to reorganize (merge) data?
    - So that everything eventually becomes a single sorted level for reads
  - Optimal = minimum work (per byte inserted) to reorganize
- Explanation and exploration in
  - Qizhong Mao, Steven Jacobs, Waleed Amjad, Vagelis Hristidis, Vassilis J. Tsotras, and Neal E. Young. 2021. Comparison and evaluation of state-of-the-art LSM merge policies. The VLDB Journal 30, 3 (May 2021), 361–378. https://doi.org/10.1007/s00778-020-00638-1

Does the index fit in memory? (5-minute-rule-type analysis)

[Appendix A of Russell Sears et al. 2012. bLSM: a general purpose log structured merge tree. ACM SIGMOD'12. https://dl.acm.org/doi/pdf/10.1145/2213836.2213862]

- What are the factors that affect this?
  - Consider kv-pair hotness and device speed
- Consider a B+-Tree index
- P: device, memory page size
  - Assume the same for now
  - Makes sense to use different page sizes for memory (index), device (data)
- ks, vs, kvs: key size, value size, K+V size
- F: Fanout for B+-Tree bottom internal nodes (BINs)
  - F=pointers+keys per memory page
  - Let's say F=100
  - Assumes 1 index page points to F device pages – approximation
- S: Device IOPS
  - A device can do S random reads(seeks)/s



How much memory M should we use to index N KV pairs?

- Let's say N is number of keys we can access in an interval T
  - Limited by available IOPS in T (random accesses)
  - N/T accesses per second
- Assume all data is on the device only
  - No data caching (large datasets with random accesses)
- N KV pair accesses will require access to N devices pages
- If index fanout is F, then we need /F pages in memory M
  - Last level of index (BINs), assume previous levels negligible
- M = N/F pages for BINs
  - M = (N/F)*P bytes for the respective portion of BINs
  - Assume index BINs are used fully (locality - all pointers they contain are traversed to reach the N KV pairs)
- What if there is some "hotness" in the keys?

Key Hotness H

- Max accesses allowed by device in interval T are IOPS*T=S*T
- If some keys are hot, then they are accessed repeatedly during T
- If they are on the device, they consume IOPS and leave less room (in IOPS) for other keys to be accessed
  - Therefore, N is reduced (unique keys accessed, depending on hotness)
  - Each requires at least one access (data always on device)
- If hot keys are accessed L times in T, then N(T) = (S/L)*T
  - Define H=T/L (seconds/access) => N(H) = S * H
- How much memory do we need to index N KV pairs?
  - M(H) = N(H)*P/F = (S*H*P)/F bytes
  - Constraint: M(T) cannot exceed the index size required to index all data on a device with capacity C, so always M(T) <= C/F = Mmax (Bytes)
- Then, memory required for index is
  - M(1s hotness) = (1M * 1s/access * 4KB) / 100 = 40 MB for index
  - M(1m hotness) → M(1s) * 60 → 2.4 GB, M(5m) = 12 GB
  - Mmax=1TB/100 = 10 GB to index fully a 1 TB device

What reduces index memory?

- Smaller index when
  - Hotter items (smaller H)
  - Larger fanout F (and assume all pointers in a page are used)
  - Fewer device IOPS (S)
- P is more strange
  - We assume the same size for pages in memory and on the device
  - No reason for this, it pays to use larger pages for index (larger fanout)
  - But we also assume locality in index
- Can we assume all of the index fits in memory?
  - Depending on hotness, the difference in index DRAM can be quite large
    - From 100s MBytes to 10s GBytes per 1TB-1MIOPS device
  - Assuming some hotness → YES
  - Assuming no hotness → Maybe or probably NO
- Are we willing to spend DRAM and keep the index always in memory?
  - Probably yes
  - More dynamic? Keep all but the last level index in memory always?
- How about the future with device capacity vs. DRAM capacity trends?

5-min-rule: What questions can we answer?

[Jim Gray and Goetz Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. SIGMOD'1997. https://doi.org/10.1145/271074.271094]

- Index: we calculated that for H=T/L (generic) we need M(H) = (S*P*H)/F Bytes
- Data? The same calculation leads to M(H) = S*P*H Bytes
- Question: Place a data item on device or in memory?
  - Consider cost (not performance)
  - Device cost = IOPS, Memory cost = capacity
  - If on device it consumes IOPS to fetch it, if in memory it consumes capacity
- Therefore, there is a frequency of accesses per item where cost is equal for memory or device
  - Call this H_b, breakeven point: same cost for placing this in memory or device, but
  - If accessed more, cheaper to place in memory
  - If accessed less, cheaper to place on device
- Calculate H_b
  - We have M Bytes of memory (at C_m $/Byte) and/or S IOPS of device throughput (at C_s $/drive with D_{IOPS})
  - Cost of memory C(M) = M * C_m (page size P)
  - Cost of storage C(S) = C_s * #drives = C_s * S/ D_{IOPS}
    - S/D_{IOPS} is the number of drives required to achieve S IOPS
  - C(M) = C(S) => $C_m * M = S/D_{IOPS} * C_s$ => $C_m * (S * P * H_b) = S/D_{IOPS} * C_s$ =>

    $C_m * S * P * H_b = S/D_{IOPS} * C_s$ => $C_m * P * H_b = 1/D_{IOPS} * C_s$ => $H_b = C_s/C_m * 1/P * 1/D_{IOPS}$
  - Therefore, $\mathbf{H_b = (1/P) \times (1/D_{IOP}) \times C_s/C_m}$ $\frac{PagesPerMBofRAM}{AccessPerSecondPerDisk} \times \frac{PricePerDiskDrive}{PricePerMBofRAM}$ (1)
  - Example: H_b = 100$/drive / (4$/GB * 4 KB * 1M (accesses/sec)/drive) = 100 / 1 sec/access = 100 sec/access
    - Can we do the same calculation for the index?
- What does it mean?
  - If you access something every ~1.5mins, keep it in memory
  - If you will access something in less than ~1.5mins in the future, keep it in memory
- What is the relationship to application performance? To cache size? To cache replacement policy?

Reading

- Review
  - Slides + pointers, Your class notes
- Read
  - VAT: Asymptotic Cost Analysis for Multi-Level Key-Value Stores. Nikos Batsaras, Giorgos Saloustros, Anastasios Papagiannis, Panagiota Fatourou, and Angelos Bilas. February 2020. arXIV 2003.00103. arxiv.org/abs/2003.00103v1
  - **(optional, Section 3)** P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. 1996. The log-structured merge-tree (LSM-tree). Acta Inf. 33, 4 (1996), 351–385. https://doi.org/10.1007/s002360050048
- Next class
  - Paper discussion

5-minute-rule-type Analysis (summary)

[Appendix A of Russell Sears et al. 2012. bLSM: a general purpose log structured merge tree. ACM SIGMOD'12. https://dl.acm.org/doi/pdf/10.1145/2213836.2213862]

- Consider a B+-Tree index
  - P: device, memory page size (assume same)
    - Makes sense to use different page sizes for memory (index), device (data)
  - ks, vs, kvs: key size, value size, K+V size
  - F: Fanout for B+-Tree bottom internal nodes (BINs). F=pointers+keys per memory page. Let's say F=100.
    - Assumes 1 index page points to F device pages – approximation
  - S: IOPS = a device can do S random reads = random seeks per sec
- How much memory (in pages) do we need to index N KV pairs?
  - Assume all data is on device, no data caching (large datasets with random accesses)
  - If we want to access N KV pairs on a device, randomly, we will eventually have to access N pages
  - If the index fanout is F, then we need /F pages in memory
  - M = N/F pages and M = (N/F)*P bytes for the respective portion of the index (BINs)
    - Assume index BINs are used fully (all pointers they contain are traversed to reach the N KV pairs)
- What is N?
  - B+-Tree requires 1 random IO per key (large dataset, random accesses)
  - Max accesses allowed by the device in an interval T are S*T
  - If some keys are hot, then they are accessed repeatedly and they do not leave room for other keys to be accessed
  - Therefore, N is smaller than the dataset size (depending on hotness)
  - If hot keys are accessed L times in the interval T, then we can define H=T/L and N(T) = S*T/L
- How much memory do we need to index N KV pairs?
  - M(H) = (S*H*P)/F bytes, Also: M(T) cannot exceed the index size required to index all data on a device with capacity C, so always M(T) <= C/F = Mmax (Bytes)
- If each key is accessed once per second then H = 1 sec / access (access to key or page on device is the same when random)
  - M(1s hotness) = (1M * 1s/access * 4KB) / 100 = 40 MB for index
  - M(1m hotness) → M(1s) * 60 → 2.4 GB, M(5m) = 12 GB, Mmax=1TB/100 = 10 GB to index fully a 1 TB device