SCMを想定した行単位更新の列指向DB反映手法

塩井　隆円†　　横田　治夫†

† 東京工業大学 情報理工学院　〒152–8550 東京都目黒区大岡山 2 丁目 12–1
E-mail: †shioi@de.cs.titech.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　OLAP の効率的な実行のために列単位でデータを格納する列指向型 DBMS を利用して大量データに対する集計が行われてきた．しかし，行単位でデータを更新する OLTP に対して列ごとにデータを読み出して格納するため，リアルタイムにデータが更新されるビッグデータ分析等では，高頻度な OLTP の更新内容を即座に反映した分析が難しい．また，OLTP によって更新されるデータを揮発性のメモリ上にソートしつつ蓄積させ，まとめてディスク上に格納するため，データの揮発を防ぐために OLTP 実行毎にディスク上でログとしてファイルを格納するディスク I/O の多い非効率な OLTP を実行している．そこで，近い将来利用可能になるとされている不揮発性メモリ（SCM）の利用を想定することで，SCM 上で行指向にデータを読み出して OLTP を効率良く実行し，揮発性メモリとディスク上の列指向データに反映する手法を提案する．

キーワード　RDBMS，列指向ストレージ，不揮発性メモリ

1. はじめに

蓄積する大量のデータを格納，更新しつつ効率良く分析するために Data Warehouse（DWH）では OLTP/OLAP 双方の性能が求められている．広告に対するクリック数等を更新，集計する Google Mesa [5] は，数分単位で大量のデータをバッチで更新し，一貫性を保証した更新と高速な OLAP 問合せを実行することが可能な分散 DWH である．また，OLAP は表形式のデータに対して少数の列を集計対象とするため，DWH として列指向型 DBMS の Vertica [7, 11] 等が一般的に利用されている．列指向型 DBMS は表形式のデータを各列単位でディスクに格納する列指向型データ格納方式を採用するため，OLAP 問合せに必要な列のみをディスクから読み出すことが可能である．さらに，各列単位でデータをソートして圧縮することで圧縮効率の高いデータをメモリキャッシュ領域に効率良くディスクからキャッシュし，列指向の問合せ処理を行うことで OLAP に最適化することができる．近年では，OLAP を高速に実行するために行指向の RDBMS のインデックスとして列指向型データ格納方式は採用されてきている [8].

しかし，近年注目されるリアルタイムビッグデータ分析では，データを更新する処理である OLTP をリアルタイムに実行しつつ OLAP の実行が求められているため，行単位で実行する OLTP に対して列単位でソートして圧縮したデータをディスクに格納する列指向型 DBMS では，特定の行を更新するために列単位でディスクから読み出し，ソート，圧縮してディスクへ格納するため，OLTP を高速に実行することができない．特に，TPC-C 系のアプリケーションではテーブル内の複数の列に対して更新が実行されるため，Vertica のような列指向型 DBMS では OLTP の実行が非効率となる（節 2.）．また，OLTP/OLAP 双方に対応したインメモリ DBMS である SAP の HANA [2,4] は，メモリ上で管理するデータ量が増加することで大規模なメモリ領域を持つ高価な計算機が必要となり容易に利用できない．

そこで，本稿では STT-MRAM [3]，ReRAM [1] 等の大容量の不揮発性メモリである Storage Class Memory (SCM) の利用を想定して OLTP/OLAP をそれぞれ効率良く実行可能な行指向／列指向データを SCM/DRAM 上のそれぞれにキャッシュするディスクを利用した DBMS を提案する．また，その際に OLTP で更新されたデータを OLAP 用の列指向データに反映する方法を提案し，OLAP 用のキャッシュ領域を有効活用することで OLTP/OLAP 双方の高速化を行う．

2. 列指向型 DBMS の OLTP

OLAP に適した列指向型 DBMS [7, 9–11] は DBMS 構成として最初に提案された C-Store [11] を基に現在様々な企業によって開発が進められている．商用の列指向型 DBMS [7, 9, 10] においても，OLTP 実行の際にディスクから各列データ毎に読み出し，再度ソート，圧縮してディスク上に行データを格納しなければならないため，一時的にメモリ上で行データを保持しておき，ディスク上データとのソートをしつつ数十万行のセグメント単位でまとめて各列毎の圧縮処理，セグメント間のソート [10]，ディスクへの格納を行っている．

メモリ上のデータは計算機の電源が消えた場合に揮発してしまうため，Vertica [7] は OLTP で INSERT，DELETE した行データと Epoch をディスク上にログとしてトランザクション単位のファイルを格納することでデータが揮発した場合の復元を可能としている．INSERT されるデータはメモリ上に蓄積するトランザクション単位の数行のデータを圧縮せずにまとめてディスク上にファイルとして格納し，DELETE されるデータはディスク上に DELETE フラグとして行 ID のアレイを作成する．また，UPDATE の際には INSERT と DELETE を組み合わせるため，DELETE フラグと非圧縮状態の数行のデータとなる INSERT ログをディスク上に格納することで OLTP 実行毎にディスク I/O が発生し，さらに，メモリ上で蓄積したセ

グメントをディスクにまとめて格納した際の Epoch 以前の複数のログファイルを削除する処理が必要となることで，OLTP 実行が非効率となっている．

さらに，Vertica [7] では OLAP 性能向上のために，ユーザ定義テーブルを分割して Projection と呼ばれる単位でディスクに格納するマテリアライゼーションを行う．Projection 毎にテーブル内でソートキーの候補となる列データを決めることでソートキー単位でデータをソートし，各列データ毎にデータの特徴に合わせて様々な圧縮方法を適用している．ソート，圧縮されたデータに対しての WHERE 句等で条件を絞り込むスキャン性能が高いだけでなく，カーディナリティの低い列データをソートキーとすることで，カーディナリティの低い列に対してカーディナリティの高い列を集計する OLAP 問合せの特徴に適しているため OLAP 性能を大きく向上させることが可能となる．しかし，様々なパタンの OLAP 問合せに対応するために様々な Projection を作成するため，同じ列データを複数の Projection に格納し，OLTP によって更新された列データを複数の Projection に反映する必要がある．そのため，Projection 毎にメモリ上で管理するデータとソート処理の増加に加え，DELETE フラグと INSERT ログも格納することによってディスク I/O がさらに増えてしまう．

列指向型 DBMS の OLTP 性能を評価するために TPC-C ベンチマークを warehouse テーブルのスケールファクター 4 で実行した．本評価の実験環境では，Amazon Elastic Compute Cloud (Amazon EC2) の r3.4xlarge インスタンス (注1) の計算機 (Red Hat Enterprise Linux Server release 7.1 (Maipo), Intel(R) Xeon(R) CPU E5-2670 v2 at 2.50 GHz processor, EBS (gp2) 500 GB SSD)を用いた．比較対象としてディスク上の行指向データを利用する RDBMS の SQLite3.8.3 と，商用の列指向型 DBMS V を機能制限なしで利用できる評価版を用いた TPC-C ベンチマークの結果を表 1 に示す．表 1 では，TPC-C ベンチマークの中で実行される複数のトランザクションの中で New-Order トランザクションを 1 分間に実行できた平均実行回数である tpmC を記述している．

表 1　行指向 RDBMS と列指向 RDBMS の TPC-C の平均スコア (tpmC)

TPC-C ベンチマーク (warehouse 4)	スコア (tpmC)
SQLite (SSD)	3,033
列指向型 DBMS V	73

表 1 の結果から，TPC-C ベンチマークに限っては，行指向型 RDBMS の SQLite は列指向型 DBMS V よりも約 40 倍 OLTP 性能が良いことが分かった．また，TPC-C は UPDATE の問合せが多く，列指向型 DBMS の OLTP 実行が INSERT と DELETE ログファイルのディスク I/O によって特に非効率となるため，TPC-C のようなアプリケーションを想定すると列指向データを用いるよりも行指向データを用いて OLTP を実行する方が効率が良いと考えられる．

そこで，本稿では行指向データを用いて OLTP を実行できるような DBMS のデータ配置を提案することで，列指向型 DBMS における OLTP 性能の向上を行う．

(注1)：https://aws.amazon.com/ec2/instance-types/?nc1=h_ls

3. 関連研究

Vertica は更新されるデータを Projection 毎に圧縮してディスクに格納するために，メモリ上で VDT と呼ばれる木構造によってディスク上とメモリ上のデータのソート順を管理している．VDT はメモリ上のテーブルに格納した行データのソートキーの値を基に，Projection 毎に格納されたディスク上データのソートキーの値を検索して行データのソート順を管理するため，ディスク上のソートキーの値を予め全て読み出しておく必要があり，OLTP のためにメモリ上に読み出すデータ量が多くなってしまう．そこで，VectorWise [6,9] では PDT とメモリ上で蓄積する行データのソート順を管理する木構造を提案している．PDT では新たにメモリ上で蓄積したデータのソートキーの値を基に，ディスク上のソートキーを検索して行データを読み出し，行 ID を管理することで VDT に比べ読み出すデータ量を減らすことが可能となり，図 1 のようにディスクから読み出した行 ID と $INSERT - DELETE$ 数を PDT の内部ノードに格納することでディスク上データの行 ID から，新たに格納するデータの行 ID を計算することが可能となる．

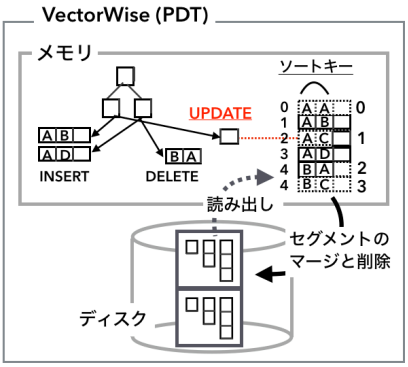

図 1　列指向型 DBMS の OLTP

しかし，Vertica と同じように揮発性メモリ上で修正したデータをディスク上に反映するために，ディスク上の行データへ DELETE フラグと，INSERT したデータをログとして格納しなければならないため OLTP 実行毎にディスク I/O が発生する．また，ソートキーの列データの値を基にソート状態から検索するスパースインデックスによってディスクから行データを読み出しており，メモリ領域の制限まで PDT が拡大した後にディスクへメモリ上のテーブルを PDT を辿りながら格納している．そして，ディスクに反映されていないメモリ上のデータを参照するために，OLAP の問合せをメモリ上の PDT を辿りながらソートキーを用いてテーブル単位で読み出すため，データ更新の頻度が高くなるほど OLAP のためにキャッシュした列単位のデータを有効活用することができない．

4. 提案手法

列指向型 DBMS では，揮発性のメモリ上に新たなデータが蓄積するため，トランザクション単位で数行のデータをディスクに格納してデータの揮発を防ぐことで，ディスク I/O の増加による非効率な OLTP が高頻度なデータ更新の際に問題となる．本稿では，近年新たなデバイス技術として着目されている大容量の不揮発性メモリである SCM（NVRAM）に着目し，SCM を有効活用するためのデータ配置によって列指向型 DBMS に比べ OLTP を効率良く実行しつつ，更新したデータを即座に反映することで，一度キャッシュしたデータ有効活用した OLAP を実行可能とすることで OLTP/OLAP 双方の高速化を行う．

4.1 OLTP/OLAP それぞれのためのキャッシュ領域

列指向型 DBMS の OLTP 実行は列指向データを用いて処理することによって各列単位で DWH のサイズが大きい SSD 等のブロックを読み出し，一行のデータを復元する必要がある．本稿では，CPU が直接ロード／ストア可能な SCM 上で行指向のデータをキャッシュしておき，行指向のデータを用いて OLTP を実行した後に行単位の更新を LRU 等を基にページ単位でディスクに書き込む方法を提案することでディスク I/O の削減を図る．また，OLTP 実行毎のログ処理を SCM 上で行うことで DRAM 上でログをまとめて書き出す場合に比べてディスク I/O の削減が可能と考えられる．

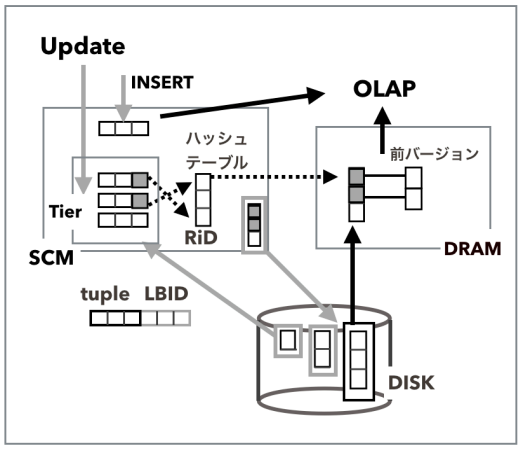

図 2　SCM 上 Tier の行データの更新

また，OLAP の効率良く実行するために，SCM 上に蓄積したデータをディスク上では列指向に格納することで，DRAM 上には必要な列データのみをロードできるようする．ディスク上では列指向型 DBMS がメモリ上に蓄積するデータの単位である数十万行単位等のセグメント毎に格納する．

そして，SCM とディスクを仮想的な Tier とみなし，DRAM は双方の Tier のキャッシュ領域として扱う．実際には，ディスク上 Tier のデータに対する更新を SCM 上 Tier の削除フラグとして利用し，ライトバックキャッシュのようにディスクに反映を行う．また，SCM 上に INSERT されるデータに対する更新はそのまま SCM 上の行指向データに更新し，SCM 上 Tier の 1 行のデータには各カラム毎にディスク上の論理ブロック ID（LBID）を持たせる．LBID を行データに持たせることでディスク上ブロックへの write の前に行 ID（RiD）からブロック ID を取得する処理を減らし，同じブロック内に属する SCM 上行データの更新をまとめる処理に利用する．SCM 上 Tier のデータに対する更新はそのまま SCM 上で行指向データを用いて実行するが，コミット時に SCM 上の RiD を基に SCM 上のハッシュテーブルから DRAM 上のカラムページの行を特定して行単位に新たなバージョンのデータをトランザクション ID タイムスタンプ（TxnID）と共にインプレイスに更新する．この時，前バージョンのデータを Versioned flag に前バージョンデータへのポインタを登録する．その後，SCM 上のハッシュテーブルに更新した行のカラム番号と LBID をディスク上のブロックへのライトバック用に登録する．DRAM 上でキャッシュした際のページバッファは図 3 のように更新する（Algorithm 1）．

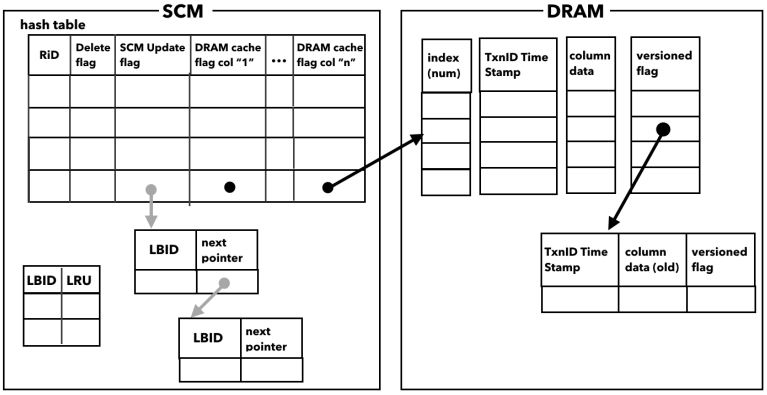

図 3　SCM 上のハッシュテーブル

Algorithm 1 DRAMcachePage

Require: $cachedPage, LBID, colnum, RiD$

```
t ⇐ hash(RiD)
t→cacheflag[colnum] ⇐ &idx[RiD]
if t→delFlag then
  pageDeleteVector append idx[RiD]
end if
t→RiD ⇐ RiD
while (uflag ⇐ t→UPDflag) ≠ NIL do
  if uflag→LBID is LBID then
    ver ⇐ create new version node
    i ⇐ idx[RiD]
    ver.TxnID ⇐ TxnID[i]
    ver.data ⇐ colData[i]
    verFlag[i] ⇐ &ver
    scmcol ⇐ scmCellRead(RiD, colnum)
    TxnID[i] ⇐ scmcol.TxnID
    colData[i] ⇐ scmcol.data
  end if
  uflag ⇐ uflag→next
end while
```

ディスク上 Tier のデータが更新される場合にはディスクから列指向のブロック単位で読み出し，SCM 上に行指向のデータ

を LBID と共に再構成して行のバージョン管理を行い，読み出された列指向のブロックは LRU によって更新頻度の低いブロックをディスク上に新たなバージョンのブロックとして書き出すことでディスク上の列指向データの更新を行う．DRAM 上では，OLAP に対して列単位でディスクからデータをキャッシュし，読み出された列指向データに対して更新があった場合に SCM 上で更新された行の RiD を基に OLTP の更新を反映する（Algorithm 2，3）．そのために SCM 上では行単位で更新したコミット時にハッシュテーブルを更新する．

Algorithm 2 SCMhashTableDELETE

Require: $currentScmTuple, updateColumnNums, delFlag$
  for $col = 0$ *to* $columns - 1$ **do**
    if $t{\rightarrow}cacheflag[col] \neq NIL$ **then**
      $i \Leftarrow t{\rightarrow}cacheflag[col]$
      $ver \Leftarrow$ *create new version node*
      $ver.TxnID \Leftarrow TxnID[i]$
      $ver.data \Leftarrow colData[i]$
      $verFlag[i] \Leftarrow \&ver$
      pageDeleteVector append i
      $LBID \Leftarrow getLBID(t{\rightarrow}RiD, col)$
      $LBIDcountLRU \Leftarrow hashLRUarr(LBID)$
      while $LBIDcountLRU \neq NIL$ **do**
        if $LBIDcountLRU.LBID$ *is* $LBID$ **then**
          $LBIDcountLRU.count \Leftarrow 0$ #コミット済みだから
        else
          $LBIDcountLRU \Leftarrow LBIDcountLRU.next$
        end if
      end while
      $LBIDcountLRU.LBID \Leftarrow LBID$
      $LBIDcountLRU.count \Leftarrow 0$
    end if
  end for

4.2 ディスクへの反映

ディスクへのページ更新を反映するために，SCM 上で更新された中で使われていない論理ブロック ID（LBID）を登録した LBID と LRU カウントのアレイを SCM 上に作成する．ディスクに格納するページにはなるべくまとめて更新後の行データを含めるために，LBID アレイには SCM 上で行単位で OLTP が実行された際のハッシュテーブルの更新時に LRU カウントを加算する（図 4）．LBID からディスク上のブロックを読み出し RiD からハッシュテーブルによって目的の行のカラムデータを取得することで更新があったデータのみを反映したブロックをディスクへ書き出ことができる．ページを書き込む際のアップデート時には，SCM にロードされていない行データが存在する可能性があるため，現状では LBID ブロックを read 後にブロック内 RiD からデータを取得し修正した後にディスクへ write している．ディスク内ブロック更新時の処理速度はカラム単位のブロックに含まれる行データの数が，通常の行指向ブロックに比べて多いため少数行の更新に関わらず多数のページに反映してしまうことで，SCM のスペース効率と処理速度が悪い点を今後改善する必要がある．

Algorithm 3 SCMhashTableUPDATE

Require: $currentScmTuple, updateColumnNums, delFlag$
  $updFlag \Leftarrow$ *create new columnupdate node*
  $col \Leftarrow updateColumnNums[0]$
  $updFlag.LBID \Leftarrow$
  $currentScmTuple.LBID[col]$
  $UpdDRAMPage(cacheMap(updFlag.LBID), updFlag.LBID, RiD)$
  $t{\rightarrow}scmUpdFlag \Leftarrow updFlag$
  $LBIDcountUpdate(updFlag.LBID)$
  if $updateColumnNums{\rightarrow}len - 1 \geqq 1$ **then**
    while col *in* $updateColumnNums$ *next* $\neq nil$ **do**
      $updcol \Leftarrow$ *create new columnupdate node*
      $updcol.LBID \Leftarrow$
      $currentScmTuple.LBID[updateColumnNums[col]]$
      $UpdDRAMPage(cacheMap(updcol.LBID), updcol.LBID, RiD)$
      $updFlag.next \Leftarrow updcol$
      $updFlag \Leftarrow updcol$
      $LBIDcountLRU \Leftarrow hashLRUarr(updcol.LBID)$
      while $LBIDcountLRU \neq NIL$ **do**
        if $LBIDcountLRU.LBID$ *is* $updcol.LBID$ **then**
          $LBIDcountLRU.count + 1$
        else
          $LBIDcountLRU \Leftarrow LBIDcountLRU.next$
        end if
      end while
      $LBIDcountLRU.LBID \Leftarrow updcol.LBID$
      $LBIDcountLRU.count + 1$
    end while
  end if

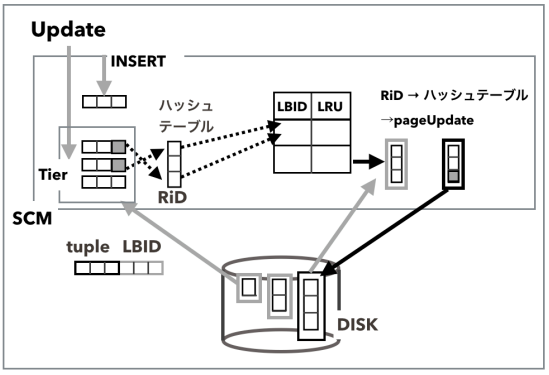

図 4　ディスクへの列データブロック write

5. おわりに

大量データの更新と集計の双方が高速に実行できることが多くのアプリケーションにとって望ましいと考えられる．集計を高速に実行できる列指向型 DBMS を利用した場合に，OLTP が非効率とされる列指向データを用いるため修正クエリの多い TPC-C のようなクエリ処理は遅く，OLAP を同時に実行した場合にも処理速度が遅くなってしまう．本稿では，OLAP/OLTP 双方の高速化を行うために列指向型 DBMS に対して SCM の利用を想定して行指向データを保持することで，行指向の効率的な OLTP 実行と DRAM，ディスク上列指向データへの反映を行う DBMS 構

成を提案した．

今後の課題として本提案手法の評価と，SCM 上の仮想的な Tier を管理する方法の考案を行う．また，SCM を実際に利用した際のデータアクセスに着目して実用的な評価を行う必要がある．そして，HTAP（Hybrid Transactional Analytical Processing）の DBMS では read set が巨大になりやすい分析系のトランザクションに対して列指向データの Scan 効率が重要になっているため，SCM のスペース効率を考慮して DRAM キャッシュに読み出す列指向データを OLTP 実行に役立てる必要があると考えられる．その際に更新のあるトランザクショナルクエリが含むカラムデータの範囲に対して Precision Locking 等を利用した validation phase の高速化等が有効と考えられる [2]．

謝　　辞

本研究の一部は JSPS 科研費 26280115 の助成を受けたものである．

文　　献

[1] H. Akinaga and H. Shima. Resistive random access memory (reram) based on metal oxides. *Proceedings of the IEEE*, 98(12):2237--2251, 2010.

[2] M. Andrei, C. Lemke, G. Radestock, R. Schulze, C. Thiel, R. Blanco, A. Meghlan, M. Sharique, S. Seifert, S. Vishnoi, et al. Sap hana adoption of non-volatile memory. *Proceedings of the VLDB Endowment*, 10(12):1754--1765, 2017.

[3] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, et al. Spin-transfer torque magnetic random access memory (stt-mram). *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 9(2):13, 2013.

[4] F. Färber, S.-K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. SAP HANA Database: Data Management for Modern Business Applications. *ACM SIGMOD Record*, 40(4):45--51, December 2011.

[5] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. Dhoot, A. Kumar, A. Agiwal, S. Bhansali, M. Hong, J. Cameron, M. Siddiqi, D. Jones, J. Shute, A. Gubarev, S. Venkataraman, and D. Agrawal. Mesa: Geo-replicated, near real-time, scalable data warehousing. In *VLDB*, 2014.

[6] S. Héman, M. Zukowski, N. J. Nes, L. Sidirourgos, and P. Boncz. Positional update handling in column stores. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 543--554. ACM, 2010.

[7] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The vertica analytic database: C-store 7 years later. *Proceedings of the VLDB Endowment*, 5(12):1790--1801, 2012.

[8] P. U. Larson, C. Clinciu, E. N. Hanson, A. Oks, S. L. Price, S. Rangarajan, and Q. Zhou. Sql server column store index. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1177--1184. ACM, June 2011.

[9] B. Răducanu, P. Boncz, and M. Zukowski. Micro adaptivity in vectorwise. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1231--1242. ACM, June 2013.

[10] A. Skidanov, A. J. Papito, and A. Prout. A column store engine for real-time streaming analytics. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1287--1297. IEEE, May 2016.

[11] M. Stonebraker, D. J. Asadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-Store: A Column-Oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 553--564. VLDB Endowment, August 2005.