巨大表形式データの高速処理を実現する自然数インデックスの定式化

古庄 晋二† 飯沢 篤志‡ 長尾 正‡† 山本 幸生‡‡

早部 秀一† 生座本 義勝† 小林 正英† 佐藤 悠†

†エスペラントシステム 〒270-0152 千葉県流山市前平井６１
‡リコーIT ソリューションズ 〒224-0035 神奈川県横浜市都筑区新栄町 16-1
‡†Layman's Admin 〒251-0033 神奈川県、 藤沢市片瀬山 1-14-7
‡‡宇宙航空研究開発機構宇宙科学研究所 〒252-5210 神奈川県相模原市中央区由野台 3-1-1

E-mail: †{s-furusho, hayabe, ozamoto, ysato}@ess-g.com, m-kobayashi@uni-network.com, ‡ atsushi.iizawa@jp.ricoh.com, ‡†tadashi.nagao@gmail.com, ‡‡yamamoto.yukio@jaxa.jp

あらまし 近年増大している巨大表形式データの処理・公開を実現するために自然数インデックスを提案している。自然数インデックスは表形式データの任意のカラム、任意のレコードに対して高速処理に寄与する機構であり、O(n)でソートが行え、表形式データが巨大なときでも高速性を維持できるという特長を持つ。本稿では、自然数インデックスを数学的に定式化する。定式化の例としてカウンティングソートとソート済みデータの仮想マージのアルゴリズムを示す。

キーワード 自然数インデックス、表形式データ、カウンティングソート、インデックス演算子

1. はじめに

近年、テレメトリデータやIoTデータなど多様な巨大表形式データが蓄積されている。そこでこれらを「表計算の感覚で利用する方法」、これらを「インターネットコンテンツの感覚で入手する方法」が求められている。ここで「表計算の感覚で利用する方法」とは、表形式データが巨大であっても、検索・集計・ソート・関係代数演算・更新などの操作を、どのカラム、どのカラムの取り合わせ、どのレコード群に対して行うのか予め決めずに対話型で操作可能にする方法である。「インターネットコンテンツの感覚で入手する方法」とは、インターネット上の巨大表形式データ群から所望の表形式データを自由に選びネットワーク上で組合せて仮想的な巨大表形式データを生成し、それを検索・集計・ソートして必要な部分を特定しダウンロードする操作を対話型で可能にする方法である。どちらも対話型であるため、巨大表形式データに対する操作が、人間の待てる時間（ほとんどが数秒以内、長くても数十秒以内）で完結することが求められる。「表計算の感覚で利用する方法」により、利用者は表形式データを入手と同時に利用でき、またそれらを容易に機械学習などの処理に引き継げる。「インターネットコンテンツの感覚で入手する方法」により、様々な巨大表形式データがインターネット上のコンテンツとして提供されるようになる。

我々はそれらを実現できる自然数インデックス[1] (NNI: Natural Numbered Index) を提案している[1]。NNI は以下の原理を用いる。

1. 任意の表形式データはその表形式データと 1 対 1 対応の関係にある成分群に分解できる。
2. 表形式データに対する検索・集計・ソート・関係代数演算などの操作に 1 対 1 対応する成分群の操作が存在する。

従って元の表形式データの操作に代わって成分群を操作することで同義の処理を行うことができる。成分群の操作が高速であることを利用したインデックスが NNI である。NNI はどのカラムにも有効なので表形式データの利用方法を予め固定する必要が無くなり、表形式データの自由な利用を実現できる。

1-1. NNI の全形態に共通する一般的特長

本稿では冒頭に掲げた「表計算の感覚で利用する方法」、「インターネットコンテンツの感覚で入手する方法」に関する 2 種類の NNI の形態を紹介する。いずれの形態にも共通する特徴を以下に挙げる。

1. カラム指向性
2. カラム対称性：どのカラムにも等しく有効。
3. 多機能性：多様なアルゴリズムが実現できる。

[1] 表形式データを自然数にマッピングすることで処理の高速化を達成することから自然数インデックスと命名した。

4. 直接参照性：NNI上の多くのアルゴリズムが1次元配列間の直接的な参照で実行でき高速である。例：$A[B[C[i]]]$

1-2. 巨大表形式データを「表計算の感覚で利用」するための Zap-In[2]

従来のインデックスは全てのカラムに張ることが難しいし、検索やソートの結果に対しては無効である。そのため表形式データのどの部分にどのようにアクセスするか予め想定できない表計算のような対話型処理には適用が難しかった。NNIの第1の形態である表形式データの成分群をメモリ上に展開する Zap-In は先に述べた NNI の 4 つの一般的特長に加え、任意のカラムの組み合わせにも有効、任意の部分集合（レコード群）にも有効、という特徴を持ち巨大表形式データの関係代数演算付き表計算を実現可能にする [1]。その特長をまとめる。

1. レコード集合対称性：任意の部分集合（レコード群）での検索やソートなどの処理を高速化できる。
2. レコードの順序保存性（安定性）：ソート以外の処理にも安定性がある。例えば年齢でソートした後、女性を検索して得られた検索結果は年齢順を保持する。安定性により検索とソートのような複合処理を単一の処理のカスケードに、複数カラムの処理を単一カラムの処理のカスケードに還元できる。そのためNNI上のアルゴリズムは任意のカラムの組み合わせに適用できる。
3. 即時利用性：CSVの読み込みや集計の結果として表形式データが生成されると同時に検索やソートなどを高速化できる。

文献[1]で示した Zap-In のアルゴリズム群から上記の3つの特長を確認でき、そこに記載されたベンチマークテストから Zap-In の高速性を確認できる。

1-3. 巨大表形式データを「インターネットコンテンツの感覚で入手」するための D5A[3]

NNI の第 2 の形態である D5A は表形式データの成分群をファイルに格納して保持する。手元にダウンロードすることが困難な巨大表形式データの公開、組合せ、検索・集計・ソート、抽出・ダウンロードをネットワーク上で実行可能にする[2]。データの公開は提供者が巨大表形式データを D5A に格納し、NAS やファイルサーバに置くだけで実施できる。データの組み合わせは利用者がネットワーク上にある単一～多数の D5A を選び、それらをネットワーク上で組合せて仮想巨大表形式データ（D5AVU[4]）を作成することで実施できる。D5AVU は任意のカラムに対して高速な検索・集計・ソートが出来る、組み合わせたデータの必要部分を抽出・ダウンロードすることが可能になる。

1-4. 定式化の目的

このように NNI は従来のインデックスとは異なる優れた特長を持つ。しかし様々な成分を様々に組み合わせて使うので理解が難しい。そこで本稿では成分群を分類した上で成分間の演算の多くを定式化できるインデックス演算を導入し理解を容易にする。

本稿では第 2 章で配列の種類とインデックス演算子を説明し、第 3 章で Zap-In の成分分解、第 4 章で Zap-In 上のカウンティングソート、第 5 章で D5A の成分分解、第 6 章で D5A を仮想マージした D5AVU を紹介する。第 7 章で Zap-In、D5A の実装、第 8 章で関連研究を述べ、第 9 章でまとめる。

2. 配列の種類とインデックス演算子

表形式データはR個のレコードとQ個のカラムからなるデータ構造である。i 番目のカラムはK^i種類の値を持ち、iが自明のときK^iをKと略記する。

2-1. 完全連番 N

0 から N-1 まで連続した自然数を完全連番 N と呼ぶ。完全連番 N の要素 i を与えると、①値の全種類数(=N)、② i より小さい値の種類数(= i)、③ i より大きい値の種類数(=N-1-i) も判る。

2-2. 配列の種類

NNI の成分は全てベース 0 の 1 次元配列である。配列の要素は整数、浮動小数点、文字列など多様である。配列の要素の型を明示したいとき、自然数配列、文字列配列などと呼ぶ。NNI の成分の多くは完全連番 N を要素とする配列であり、これを完全連番N配列と呼ぶ。以下にこれらの配列の記法をまとめる。

- 配列 Aのサイズ n を明示するときは$A_{(n)}$と書く。
- 配列 $A_{(n)}$の i 番目の要素は$A_{(n)}[i]$と書く。
  $A_{(n)} \equiv (A_{(n)}[0], A_{(n)}[1], \ldots, A_{(n)}[n-1])$ である。
- 配列 A の要素が完全連番 N であることを明示したいときは $A^{(N)}$ と書く。

2-3. 対称配列、非減少配列、増加配列

サイズ N の配列 P が完全連番 N の値を重複せずに持つ（$i \neq j \Rightarrow P^{(N)}_{(N)}[i] \neq P^{(N)}_{(N)}[j]$ ）とき、対称配列と呼ぶ。

---

[2] Zap-In は成分群を収容するインメモリのデータ構造を指す。**Zap In** Memory Tables の略。

[3] D5A は成分群を収容したファイルを指す。文献[2]では、Zap-Over と呼んでいたが、今回 D5A という名称に変更する。D5A という名前は「**第五**世代の**ア**ーカイブ」から作られた。

[4] D5AVU(D5A Virtual Union) は D5A を組み合わせた仮想巨大表形式データであり、D5A の組合せ方を記述したスクリプトファイルで定義される。

また、$i<j \Rightarrow A^{(N)}[i] \leq A^{(N)}[j]$ である A を非減少配列と呼ぶ。予め $A[-1] \equiv 0$ と決めておく。

さらに、$i<j \Rightarrow A^{(N)}[i] < A^{(N)}[j]$ である非減少配列 A を増加配列と呼ぶ。

2-4. インデックス演算子[5]

NNI の成分はしばしば以下で定義するインデックス演算子(Index Operator) ∘ で組み合わせて使用される。

$$A_{(n)} \circ B^{(n)}_{(m)} = \left(A_{(n)}\left[B^{(n)}_{(m)}[0]\right], \ldots, A_{(n)}\left[B^{(n)}_{(m)}[m-1]\right] \right)$$

インデックス演算子には以下の性質がある。

- 結合則が成り立つ
  $(A_{(l)} \circ B_{(m)}) \circ C_{(n)} = A_{(l)} \circ (B_{(m)} \circ C_{(n)})$
- 演算結果のサイズは右側の配列のサイズ
- 演算結果のデータ型は左側の配列のデータ型

対称配列はインデックス演算子に関して群を作る。

単位元：$E = (0, 1, \ldots, N-1)$

逆元：$P[i] = j \Rightarrow P^{-1}[j] = i$

図 1. 配列の種類

3. Zap-In の成分分解とその利用法

図 2 で Zap-In の成分分解とその 3 つの成分 SVL, NNC, S を示す。SVL と NNC はカラムに関する成分であり、S はレコードに関する成分である。S はインデックス演算子を用いてさらに L と P に分解できる。

3-1. カラムに関する成分：SVL[6]、NNC[7]

1 つのカラムは R 個の値を保持する非自然数配列 $C_{(R)}$ と見なせる。$C_{(R)}$ に含まれる K 種類の値を取り出し昇順に格納すると昇順値リスト成分 $SVL_{(K)}$ が得られる。つぎに $C_{(R)}$ の各要素を SVL 上の格納位置で置き換えると完全連番 K 配列である自然数化項目値成分 NNC ができる。$C_{(R)} = SVL_{(K)} \circ NNC^{(K)}_{(R)}$ である。

例）$C_{(6)} = (\text{Bob}, \text{Cathy}, \text{Alice}, \text{Bob}, \text{Bob}, \text{Cathy}) \Rightarrow$

$$SVL_{(3)} = (\text{Alice}, \text{Bob}, \text{Cathy}), NNC^{(3)}_{(6)} = (1, 2, 0, 1, 1, 2)$$

つまりカラムは SVL と NNC に変換される。

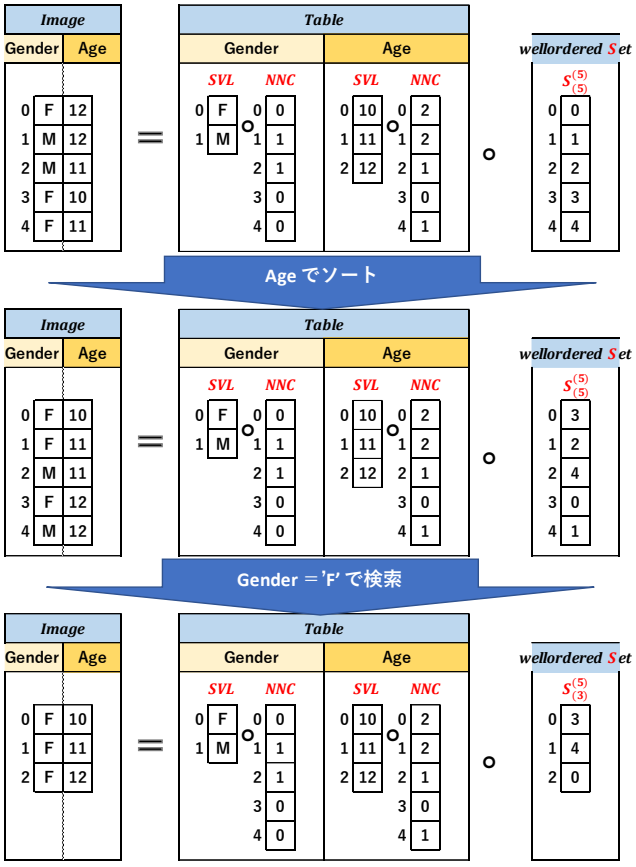

図 2. Zap-In の成分分解とソートと検索における成分の変化

3-2. レコードに関する成分：S[8]

表形式データのレコードに、先頭から順に 0...R-1 のレコード識別子を与える。レコード識別子を重複無く格納したサイズが n(≤R) の完全連番 R 配列 $S^{(R)}_{(n)}$ $(0 \leq n \leq R, i \neq j \Rightarrow S^{(R)}_{(n)}[i] \neq S^{(R)}_{(n)}[j])$ を使って表形式データに対する検索とソートのあらゆる結果を表せる。

例えば、図 2 下段の $S^{(5)}_{(3)} = (3,4,0)$ は 5 個のレコードから 3,4,0 番目を選んだことを示している。この S を順序付きレコード選択成分と呼ぶ。

なお、$S^{(R)}_{(R)}[i] = i$ である $S^{(R)}_{(R)}$ は元々のレコード順に並んだ全体集合を表しており、これをルート集合と呼ぶ。ルート集合は CSV などの外部の表形式データを成分分解した際に作られる。

ここで図 2 の上段から下段にかけて S の変化を見てみよう。図 2 ではソートと検索しか行われておらず、データの更新がないのでカラムに関する成分 SVL と

[5] インデックスは添字という意味
[6] **S**orted **V**alue **L**ist
[7] **N**atural **N**umbered **C**olumn
[8] well-ordered **S**et

NNC は変化せず、レコードに関する成分である S のみが変化する。上段で S はルート集合である。中段の S は Age でソートされた結果集合である。Gender で検索された後の下段の S は検索により $(3,\cancel{2},4,0,\cancel{1}) \mapsto (3,4,0)$ と変化するが、検索の安定性[9]により順序関係は維持される。

このように検索やソートを S のみの変化で記述できることは Zap-In の高速性の理由の一つである。

3-3. S の L[10]、P[11]への分解

S は $S_{(n)}^{(R)} = L_{(n)}^{(R)} \circ P_{(n)}^{(n)}$ としてレコード選択成分 L とレコード順序成分 P に分解できる。L は増加配列、P は対称配列である。例えば S = (3,4,0) ならば、L= (0,3,4) で、P = (1,2,0) になる。このとき P は対称配列であるから逆元 P^{-1} = (2,0,1) を求めることができる。これを利用すると 1 つの表形式データを別々のカラムでソートして得られた 2 つの表形式データ間でのデータのやりとりが容易になる。

3-4. データの読み出し、検索、ソート[12]

Zap-In からデータの読み出し、検索、ソートをどう行うのか以下に示す。

1. データの読み出し

あるカラムの S に関する i 行目のデータは、

$\mathrm{SVL}_{(K)} \circ \mathrm{NNC}_{(R)}^{(K)} \circ S^{(R)}[i]$ で与えられる。

例えば図 2 の下段の Age の 1 行目は以下で求まる。

$\mathrm{SVL}_{(3)} \circ \mathrm{NNC}_{(5)}^{(3)} \circ S_{(3)}^{(5)}[1] = \mathrm{SVL}_{(3)} \circ \mathrm{NNC}_{(5)}^{(3)}[4] = \mathrm{SVL}_{(3)}[1] = 11$

2. 検索

検索前の S を $S_{(l)}^{0}$、検索後を $S_{(m)}^{1}$ とする。

$S_{(m)}^{1} = \left(S_{(l)}^{0}[i_0], S_{(l)}^{0}[i_1], \ldots, S_{(l)}^{0}[i_{m-1}] \right)$ が検索結果である。

$(i_0, \ldots, i_{m-1}$ は $\mathrm{SVL}_{(K)} \circ \mathrm{NNC}_{(R)}^{(K)} \circ S_{(l)}^{0\,(R)}[i]$ が検索条件を満たす $i)$

例えば図 2 の中段から下段にかけての検索結果は

$S_{(m)}^{1} = (3,4,0)$ で、$\mathrm{SVL}_{(K)} \circ \mathrm{NNC}_{(R)}^{(K)} \circ (3,4,0) = (F,F,F)$

Zap-In の検索は順序を変えない。これをソートの安定性と同じ意味で検索の安定性と呼ぶ。

3. ソート

カウンティングソートを用いる。その詳細は次章で説明するがカウンティングソートには安定性がある。

3-5. Zap-In のまとめ

Zap-In は任意のカラムに対して高速な検索やソートが行える。また検索やソートに安定性があり、複数カラムの検索やソートを 1 カラムの検索やソートに還元できる。これにより任意のカラムの組合せに対して効率よく検索やソートが可能になる。

Zap-In の成分分解から D5A の成分分解へと進む前にカウンティングソートを説明しておく必要がある。

4. カウンティングソート

カウンティングソートは $O(n)$ と効率が高いソートであるが、ソート対象が整数かつ限られた範囲の値である必要があり、利用しにくい [3]。ところが Zap-In では NNC が完全連番 K 配列であるため、常にカウンティングソートが利用できる。このカウンティングソートには安定性があることが知られている。安定性のある処理は組合せが容易で、図 2 で示したように検索とソートを組み合わせることもできる。

図 3 は図 2 の Age カラムをカウンティングソートする際の 3 つのステップを示している。

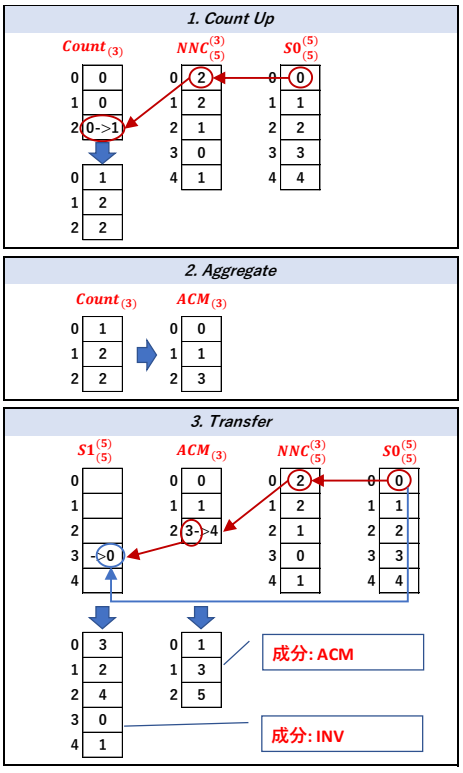

図 3. カウンティングソートの 3 ステップ

1. Count Up ステップでは $NNC \circ S$ で得られる完全連番 K 配列：$Count$ の要素の出現度数を数える。図 3 では $NNC \circ S = (2,2,1,0,1)$ で、$Count = (1,2,2)$ となる。

2. Aggregate ステップでは、$i = 0$ のとき $ACM[0] = 0, i > 0$ のとき $ACM[i] = \sum_{j=0}^{i-1} Count[j]$ と設定する。

[9] 3-4.2 で説明がある

[10] seLection

[11] Permutation

[12] 集計は文献 [1] に説明がある。

$Count = (1,2,2)$なので$ACM = (0,1,3)$ となる。

3. Transfer ステップでは以下を行う。

$for\ i := 0\ to\ (n-1)\ do\ begin$

$$S1_{(n)}^{(R)}\left[ACM_{(K)}\left[NNC_{(R)}^{(K)}\left[S0_{(n)}^{(R)}[i]\right]\right]\right] := S0_{(n)}^{(R)}[i];$$

$$increment(ACM_{(K)}\left[NNC_{(R)}^{(K)}\left[S0_{(n)}^{(R)}[i]\right]\right]);$$

$end;$

上記のカウンティングソートの Count Up ステップおよび Aggregate ステップは Transfer ステップでのソート結果の格納先アドレスを計算するための準備である。カウンティングソートを適切に改造すると 6 章で述べる D5AVU（ソート済みデータの仮想マージ）が実現できる。

5. D5A の成分分解とその利用法

D5A の成分分解は Zap-In の成分分解から S が除かれ図 3 中の ACM と INV が加えられたものである。図 4 に図 2 の Age カラムに対する D5A の成分分解を図示する。

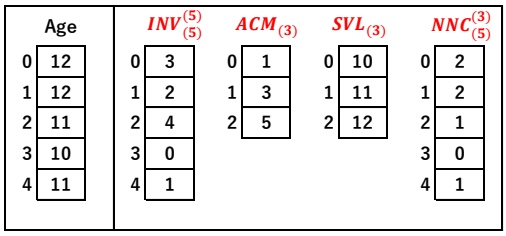

図 4. D5A の成分分解：Age カラムの INV, ACM, SVL, NNC

5-1. ACM[13]

図 3 中、ソート後に得られる $ACM[i]$ は $SVL[i]$ 以下の値の出現度数を表し、ACM は D5A 中の重み累計成分になる。$ACM[-1] \equiv 0$ として以下の式(1)～(3)が成り立つ。

$SVL[i]$以下の値の個数： $ACM[i]$ ...式(1)
$SVL[i]$より小さい値の個数：$ACM[i-1]$ ...式(2)
$SVL[i]$に等しい値の個数： $ACM[i] - ACM[i-1]$ ...(3)

5-2. INV[14]

図 3 中、$S0$、$S1$はソート前と後の順序付きレコード選択成分である。$S0$がルート集合のとき、$S0$をソートした$S1$は転置インデックス成分 INV になる。

5-3. INV,ACM と NNC,S の関係

カウンティングソートによって NNC と S から INV と ACM を生成した。逆に ACM と INV から$NNC \circ S$を生成できることを示しておこう。特にSがルート集合（＝単位元）ならばNNC を生成できる。（図 5）

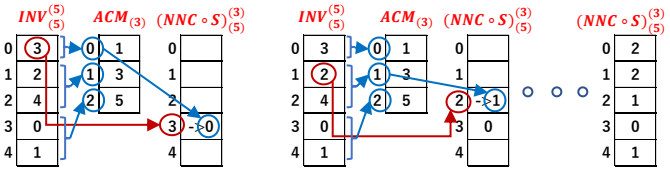

図 5. ACM と INV から NNC を生成する処理

5-4. データの読み出し、ソート、集計、検索

以下の式(4)を使ってデータの読み出し、式(5)を使ってソート、式(3)を使って各値の出現度数の集計ができる[15]。これらは O(1)で完了し高速である。

i 行目のデータ取得： $SVL \circ NNC[i]$ ...式(4)
i 行目のソート結果： $SVL \circ NNC \circ INV[i]$ ...式(5)
SVL 中の i～j 番目の値を保持するレコード番号：
$(INV[ACM[i-1]], \ldots, INV[ACM[j]-1])$ ...式(6)

検索の方法を説明する。値 x～y の検索は 2 段階で行う。初段では x, y に対応する SVL 上の位置 i, j をバイセクション法などを用いて特定する。検索の初段は O(logK)のステップ数を要する。検索の後段では検索結果集合を作成する。検索結果集合は O(1)のステップ数しか要さない式(6)を用いて高速に作成される。

5-5. D5A の成分分解のまとめ

D5A はローカル、ネットワークの区別無く動作し、データの読み出し、ソート、集計、検索が短時間で完了することを 5-1～5-4 で示した。D5A のこれらの機能を土台にして次章で D5AVU にも同様の機能が実現できることを示す。Zap-In と D5A の成分の一覧を表 1 にまとめる。

表 1. レコードとカラムに関する成分の一覧

記号	成分名	配列種類、特長
レコード関連（Zap-In のみ）		
S	順序付きレコード選択成分（初期状態をルート集合と呼ぶ）	完全連番 R 配列 重複要素を持たない $L \circ P$ に分解できる
L	レコード選択成分	完全連番 R 配列 増加配列
P	レコード順序成分	完全連番配列 対称配列
カラム関連（Zap-In と D5A 共通）		
SVL	昇順値リスト成分	非自然数配列 重複要素を持たない 要素が昇順に並ぶ
NNC	自然数化項目値成分	完全連番 K 配列
転置レコード関連（D5A のみ）		
INV	転置インデックス成分	完全連番 R 配列 対称配列
ACM	重み累計成分	非減少配列

[13] **AC**cu**M**ulation
[14] **INV**erted record numbers
[15] D5A は巨大表形式データを収容するので、データの読み出しはもちろん、ソート、集計、検索についても取得したい部分を指定してそこだけを取得する。

6. D5AVU とその利用法

本章では単数～多数の D5A を縦方向に仮想的にマージして生成した仮想的な巨大表形式データである D5AVU が高速なデータ取得、ソート、集計、検索を行えることを示す。ただし、ここでは紙幅の関係でそれぞれ指定のアドレス 1 個の結果の取得方法を示す。例えばソートではソート結果の i 番目のレコードのみの取得方法を示す。効率的で一般性がある範囲の取得方法は論文を改め丁寧に議論する必要がある。

図 6 は $D5A^0$, $D5A^1$ を縦方向に仮想的にマージし、D5AVU を作成した状態を表す。$T, Sorted\ T$ はそれぞれレコード順ビュー、ソート順ビューである。なお図中で NNC は省略した。成分から合成される仮想的なデータ[16]は点線で囲んだ。$D5A^0$ に由来するデータは黒文字、$D5A^1$ に由来するデータは赤文字で表し、両方に由来するデータは黒文字で始め、赤文字で終わらせている。また Alice に対応するセルは水色、Beth は薄緑、Bob は薄橙、Cathy は薄赤で塗りつぶした。赤文字で示した $INV^{Virtual}$ 中の $D5A^1$ に由来するデータは $R^0(=5)$ を加算して読み替えている。

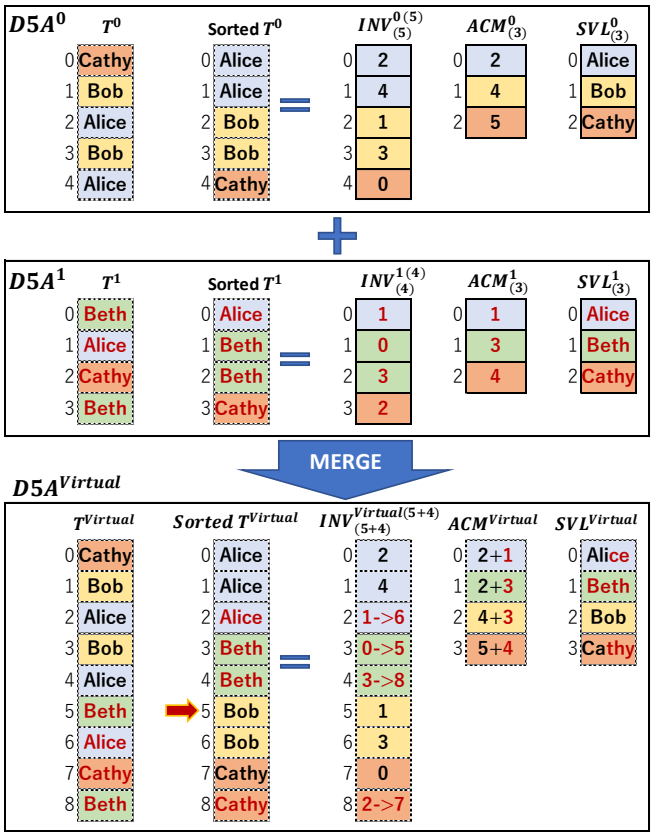

図 4. ソート済みデータ(T^0 と T^1)の仮想マージ

6-1. データ取得

レコード順ビューである $T^{Virtual}$ のレコード番号 i のデータ取得のアルゴリズムを述べる。最初に T^0, T^1 のレコード数 $R^0=5$, $R^1=4$ を使い i が T^0, T^1 のいずれに属するかを判別する。その後、i を i が属する表形式データ内のレコード番号に変換しデータを取得する[17]。

例えば i=5 のとき $i \geq R^0$ であるから、T^1 に属することが判る。$i-R^0=5-5=0$ であることから $T^{Virtual}[5]=T^1[5-5]=T^1[0]=Beth$ を得る。

6-2. 集計と検索

D5AVU の集計は個々の D5A の集計結果を併合することで実現できる。例えば Alice は T^0 に 2 個、T^1 に 1 個あり、合計 3 個あることは簡単に判る。

検索は 5 章で述べた 1 つの D5A に対する検索アルゴリズムを各 D5A に対して行えば良い。各 D5A の検索は高速なので D5AVU の検索も高速に実行できる。

6-3. ソート

例としてソート順ビュー $Sorted\ T^{Virtual}$ の 5 行目の取得方法を示す。

まず SVL^0、SVL^1 の中から適当な値 v を選び、式(1)～(3)を使って 5 行目が v より小さい値に属するか、v に属するか、v より大きい値に属するか、を調べる。v に属さない場合は新たな v を 5 行目に近づくように選び直す。これを繰り返すと 5 行目がどの値に属するかが判る。図 6 の場合、Bob に属することが判る。

次に同じく式(1)～(3)を使ってその値 Bob が T^0、T^1 のいずれに属する Bob なのかも判る。図 6 では T^0 に属することが判る。

最後に同じく式(1)～(3)を使ってその値 Bob が T^0 中の何番目の Bob なのかも判る。図 6 では 0 番目の Bob であることが判る。こうして T^0 中の 0 番目の Bob に該当するレコード番号 $INV^{0(5)}_{(5)}[2]=1$ が特定できる。T^0 のレコード番号 1 とまで判れば全てのカラムの値を取得できることになる。

この方法は全データをソートするわけでなく指定された行のみ（この例では 5 行目のみ）をソートする方法であり、R^0, R^1 の大きさに拠らずおよそ $O(\log(K^0)+\log(K^1))$ で完了できる[4]。

6-4. D5A を仮想マージした D5AVU のまとめ

D5AVU は単数～多数の D5A をネットワーク上で縦方向に仮想的にマージしたものとして生成される。D5AVU で、巨大表形式データの入手のための 4 つの基本操作（公開、組合せ、検索・集計・ソート、抽

[16] D5A に格納された成分 INV, ACM, SVL 以外は全て仮想的なデータである。

[17] このとき R^0, R^1,… を予めメモリ上に保持しておく。

出・ダウンロード）がネットワーク上で行える。

7. Zap-In・D5A の実装例

Zap-In の実装例[18]では 10 億レコード、1 千カラムまでの表形式データを処理でき、任意のカラム、任意のカラムの取り合わせ、任意の部分集合（レコード群）に対する処理が高速である。検索に限らず、集計・ソート・関係代数演算・一括更新・抽出・ユニオンなどの処理も高速である。一方、複数のユーザが同一データにアクセスできる仕組みは備えていない。つまり巨大表形式データを関係代数演算付きの表計算の感覚で利用することが実現できる。また Zap-In は様々なバッチ処理を高速に実行する基盤としても利用できる。

一方、D5A の開発中の実装例[19]では 1 兆レコード、10 万カラムまでを格納できる。D5A を仮想的に組み合わせた D5AVU は最大 100 個までの D5A を自由に選び組み合わせることができる。Apollo11 号～17 号が月面に設置した地震計のデータを格納した 40 個の NAS 上の D5A を LAN 上で組合せ、1350 億レコードの D5AVU を作成し、検索やソートができた。その際、D5AVU の作成、検索、ソートのいずれも 1 秒未満であった。

8. 関連研究

NNI では、値の集合を固定長の 1 次元配列で表現し、値と 1 次元配列の添字を 1 対 1 で関連付ける。値あるいは値に関連する情報を 1 次元配列に格納する手法はコンピュータの初期から利用されているデータ構造である[5]。添字から配列の値を得るコストは O(1) で効率的なので、0 からある範囲の自然数で対象を特定することは良く行われる。NNI では、レコード ID を、各カラムの値をもつ 1 次元配列 NNC の添字として使い、そのレコードのカラムの値を得る。また、レコード ID を値にもつ 1 次元配列*S*を使い、ソートや検索の結果を表現する。

このように 1 次元配列を使って定義域の型から値域の型へ変換する写像を表現することができる[6]。SVL はソートされた値の 1 次元配列であり、完全連番から値の集合への全単射写像である。SVL により、値から自然数に順序関係を保って変換することができる。ソートされた値の列から値を検索するコストは O(logN) であり、一度検索して自然数に変換すると、そのあとは、自然数内の操作だけで、ソート・検索など種々のアルゴリズムが実行できる。NNI で 1 次元配列を使って高速アルゴリズムを実現できる背景には、近年のコンピュータの高速化、メモリ容量の増大が寄与している。

このような NNI の基本的アイデア自体は古く、普通に使われている手法であるが、NNI は、それをすべての値集合に適用し、完全連番に変換することで、統一的に種々のアルゴリズムを実現する。

データを高速にアクセスする構造・手法は古くから数多く考案されてきた。値の追加・削除・更新があるときに、1 次元配列の状態で値をソートされた状態に保つのはコストが高いので、値の変化に応じて効率よく更新するためにリスト構造やツリー構造など様々なデータ構造が古くから考案されている[5]。DBMS で使うファイル構造としては、ツリーの高さのバランスが保たれ、値を昇順・降順に辿ることができる B+ツリーがもっとも普及している[7]。値から対応する情報に高速にアクセスする方法としてハッシュ表もよく利用される[5]。ハッシュ表は、値の順に辿ることはできないため、範囲検索などを実現するには別のデータ構造を併用する必要がある。

NNI はカラム指向の高速化機構であり、データベースではないが、カラム指向ではあるデータベースと比較するのは有意義である。

HBase[8]、Cassandra[9]は、列ファミリーデータストアと呼ばれる[7]。列ファミリという関連する列が隣接領域に格納され、行キーに対する列の値をまとめてアクセスすることができる。列の値からのアクセスを高速化するために二次インデックスを設定できる。

Sybase-IQ[10]は、ランダムアクセスできないので性能低下を補うために 9 種類のインデックスが搭載されている。またカラムの中間に新データを挿入するのに時間がかかるため、新データの中間挿入は禁止し、末尾への追加のみに制限している。

C-Store[11]は、大規模データ対応のためシェアドナッシング方式の超並列方式を導入している。このため機能的な制約が大きく、多数のユーザの同時使用はしにくい。また、上記と同じく新データは末尾に追加のみに制限している。

SAP-HANA[12]はインメモリであり、原則的としてインデックスは使わない。インメモリを活かして新データの中間挿入ができるが、1 つのカラムを多数の小ブロックに分割して格納することで、中間挿入の速度を上げて OLTP に対応している。しかしこのため、データベースの大域を扱う処理での速度低下が激しい。

NNI は、カラムごとに成分分解してデータを保持し、レコード番号に対応する値をアクセスする、そして二次インデックスを設定することなく、各カラムの値からあるいは任意のカラムの組合せからも高速にア

[18] エスペラントシステム社の Super DataChef もしくは Esperic

[19] エスペラントシステム社の D5AReader

クセスできる機構である。

9. まとめ

近年増加しているテレメトリデータやIoTデータなど多様な巨大表形式データを表計算の感覚で利用したり、インターネットコンテンツの感覚で入手したりするために、我々は自然数インデックス (NNI: Natural Numbered Index) を提案している。NNI は表形式データを所定の成分群に分解し、効率の良い成分群の操作で検索・集計・ソートなどを行うことで高速化を実現する。本稿では Zap-In、D5A の 2 種類の NNI 実装形態を紹介した。Zap-In はインメモリであり「表計算の感覚で利用する方法」を実現できる。D5A はオンディスクであり「インターネットコンテンツの感覚で入手する方法」を実現できる。

NNI の理解を助ける定式化として、成分群を分類し、インデックス演算子を導入して成分間の演算を簡潔に説明できるようにした上で、まず、Zap-In という NNI の実装形態を用い巨大表形式データを「表計算の感覚で利用する方法」を示した。Zap-In は任意のカラム、任意のカラムの取り合わせ、任意の部分集合に対する処理を高速化でき、5G の普及などで多様な表形式データが入手できるようになった時代に必要になる技術である。

つぎに、Zap-In のアルゴリズムの中で重要なカウンティングソートを説明した。カウンティングソートはまずソート対象データを値毎に分離し、それらを値の順番に結合するソートである。カウンティングソートがソートの安定性を持つのは前記の結合の際に同一値の中に隠れた順序関係を保存するためであると理解できる。次章で述べた D5A の場合の隠れた順序関係とはテーブル順およびテーブル内のレコード順である。そしてそのカウンティングソートを行うときに生成される *ACM* と *INV* の両成分が D5A の中心的構成要素になることは興味深い。

D5A を使うとネットワーク上の表形式データが任意のカラムで高速な検索・集計・ソートができる。さらにネットワーク上で仮想的に D5A を組み合わせて D5AVU が実現できる。D5AVU も D5A 同様に表形式データの任意のカラムで高速な検索・集計・ソートができ、ネットワーク上で 4 つの基本操作（公開、組合せ、検索・集計・ソート、抽出・ダウンロード）が行える。これにより、5G の時代に求められるビッグデータを「インターネットコンテンツの感覚で入手する方法」が実現できる。

ここで述べた NNI の基本的アイデアはコンピュータの黎明期からあり、コンピューティングの根源的なものであると考える。NNI はそれらをまとめ直し体系化したものであり、コンピューティングの新たな出発点となることを期待する。

謝辞

本研究は、総務省戦略的情報通信研究開発推進事業（SCOPE）の独創的な人向け特別枠「異能 vation」プログラムの支援を受けたものです。

参 考 文 献

[1] 古庄晋二,飯沢篤志,長尾正,山本幸生,早部秀一,生座本義勝,小林正英, “宇宙科学分野のビッグデータ処理を高速化する自然数インデックス (NNI: Natural Number Index)”, 宇宙科学情報解析論文誌第 10 号, pp.1-66, 2021.

[2] 古庄晋二,飯沢篤志, “ZAP-Over: インターネット上に分散した SVL によるビッグデータのブラウジング手法”, DEIM 2018,2018.

[3] D.E.Knuth, “The Art of Computer Programming VOLUME 3: Sorting and Searching”, Addison-Wesley,1973.

[4] 古庄晋二, PCT/JP2019/005141, 2019.（仮想マージソート）

[5] D.E.Knuth, “The Art of Computer Programming VOLUME 1: Fundamental Algorithms”, Addison-Wesley, 1968.

[6] A.V.Aho,et al, “Data Structures and Algorithms”, Addison-Wesley, 1983.

[7] 増永良文, “リレーショナルデータベース入門―データモデル・SQL・管理システム [第 3 版]”, サイエンス社, 2017.

[8] Apache HBase, https://hbase.apache.org/

[9] Apache Casssandra, https://cassandra.apache.org/

[10] “Sybase-IQ の概要”, http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc37422.1540/pdf/iqintro_ja.pdf

[11] M.Stonebraker, et al, “C-Store: A Column-oriented DBMS", Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005.

[12] “SAP HANA とは?”, https://www.sap.com/japan/products/hana/what-is-sap-hana.html