データ・レイアウトとスケジューリングの最適化による テープ上のストリーム・データ分析高速化

飯澤 健[1,a)]　田村 雅寿[1,b)]

概要：センサ・データやパケット・データ，ログ・データ等，時々刻々と生成されるストリーム・データを保存しておくことで，過去の事象を遡って分析することが可能になる．大容量化・高速化が進むテープは，そうした大量のストリーム・データを低コストで格納する用途に適している．しかし，テープはシーケンシャル・アクセス性能に優れる一方，ランダム・アクセス性能が低く，また，カートリッジ交換のオーバヘッドも大きい．そのため従来は，テープに保存されたストリーム・データに直接アクセスしてデータ分析を行うことは難しかった．本論文では，テープに保存されたストリーム・データへの SQL アクセスを高速化する手法を提案する．提案手法は，クエリ実行履歴の分析結果に基づき一部のカラムを元データとは別のカートリッジにコピーし，クエリ実行時にアクセスするカートリッジを選択することで，読み出し性能を向上させる．簡易的なベンチマークを用いた予備実験により，本手法による性能向上の可能性を評価する．

1. はじめに

センサ・データやパケット・データ，ログ・データ等，時々刻々と生成されるデータはストリーム・データと呼ばれる．機器が生成するストリーム・データをリアルタイムに監視することで，機器の状態監視や異常検出を行うことができる．また，ストリーム・データを一定期間保存しておくことで，過去の事象を遡って分析することが可能になる．例えば，PC が送受信するパケット・データを保存しておくことで，サイバー犯罪発生時に侵入者の侵入経路の特定等が可能になる．

IoT の普及により近年増大しているのが，センサが生み出す構造化されたストリーム・データである．例えばコネクテッド・カーは，CAN データと呼ばれるタイムスタンプと多数のセンサ値からなるデータを，定期的にデータセンターに送信する．データセンターに保存されたデータをアプリケーションが分析することにより，運転手の行動分析 [4] や車両の診断 [8]，燃料消費の効率化 [2] 等が可能になる．CAN データは構造を持つため，アプリケーションからは SQL でアクセスされるのが一般的である [8][10][12]．

CAN データを保存する際に問題となるのが，その膨大な量である．例えば，2018 年の時点ではコネクテッド・カー 1 台あたり 1TB/日のデータを送信していたとの見積もりがある [8]．HDD の記録密度向上は鈍化傾向にあり [3]，CAN データを全て HDD に格納するのは高コストとなる．低価格化が進むクラウド・ストレージも選択肢となるが，data-intensive なアプリケーションの場合は性能要件を満たすのが難しい [9]．

そこで，有力な格納先の候補となるのがテープである．テープの容量単価は今でも全ストレージ・デバイスの中で最も安く，この傾向はこれからも続くと見込まれている [3]．また，記録密度の向上にともない，シーケンシャル・アクセス性能の向上も期待できる．

ただし，テープはシーケンシャル・アクセス性能に優れる一方，ランダム・アクセス性能が低く，また，カートリッジ交換のオーバヘッドも大きい．従来のようなバックアップ用途であればシーケンシャル・アクセス性能さえ出れば問題なかった．しかし，データ・マイニングや機械学習の発展により，今後，過去のアーカイブ・データに新しい分析手法を適用することで新しい知見を得られる場面が増えてくる．そのため，アーカイブ・データにも実用的なアクセス速度でアクセスできる必要がある．

低コストで高速アクセスが可能なストレージ・システムを実現する方法として，テープと HDD を組み合わせる階層化アーキテクチャが挙げられる．これは，頻繁にアクセスされるデータを HDD に，あまりアクセスされないデータをテープに格納することで，平均アクセス性能を向上させるものである．ただし，階層化アーキテクチャは大量の

[1] 株式会社富士通研究所
Fujitsu Laboratories Ltd., Kawasaki, Kanagawa 211-0053, Japan
[a)] iizawa.ken@fujitsu.com
[b)] masahisa@fujitsu.com

HDD を使用するためコストの増大につながる．また，1 回きりのアクセスの場合には性能向上効果が見込めないという問題もある．

本論文では，テープに保存されたストリーム・データへの SQL アクセスを高速化する手法を提案する．提案手法は，ストリーム・データはカラム間のアクセス頻度に偏りがあることに着目し，データ格納時に一部のカラムを元データとは別のカートリッジにコピーして保存する．クエリ実行時には，必要なカラムを含むカートリッジの中から最も高速に読み出しができるカートリッジを選択して読み出すことにより，テープのシーケンシャル・アクセス性能を最大限に活用する．

簡易的なベンチマークを用いた予備実験により，本手法による性能向上の可能性を評価する．

本論文の構成は以下の通りである．第 2 節で，ストリーム・データをテープに保存する場合に発生する問題点を明らかにする．第 3 節で，本論文における提案手法を示す．第 4 節では簡易的なベンチマークを用いた予備実験の結果を示す．第 5 節で関連研究の紹介を行い，最後に第 6 節で今後の課題を述べまとめとする．

2. 背景

2.1 テープの性能特性

広く普及している磁気テープ技術である LTO では，ヘッドはテープ・カートリッジに対して図 1 のようなアクセスを行う．

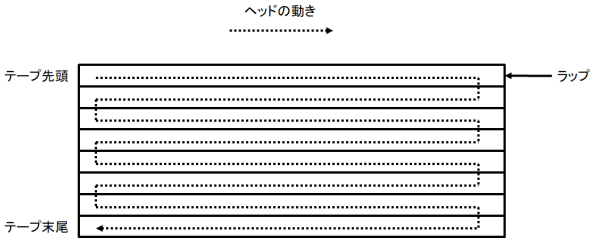

図 1　ヘッドのテープ・カートリッジへのアクセス

テープ・カートリッジはラップと呼ばれる記録領域から構成される．LTO-7 の場合ラップ数は 112 である．データをテープ・カートリッジ全体から読み出す場合，ヘッドは最初のラップの先頭から読み出しを開始し，ラップの末尾に到達するとヘッドを次のラップに移動し，テープを送る方向を反転させ読み出しを継続する．こうした機構の結果，テープの次のような性能特性が生まれる．

- ラップ内に閉じたシーケンシャル・アクセスはテープ送りの速度で実行できるため高速である．
- ラップ内に閉じたランダム・アクセスの際には，データ転送の間にシークが発生する．シークはデータ転送同様にテープ送りの速度で実行されるため低速である．
- ラップ間に跨るランダム・アクセスの際には，ヘッド移動とシークが発生する．ヘッドのラップ間移動はシークに比べ高速なため，ヘッド移動後のシーク距離が短い場合は高速なアクセスを行うことができる場合もある．

2.2 従来技術の問題点

ストリーム・データとして，図 2 のようなデータを想定する．

時刻	カラム1	カラム2	…	カラムj
t_1	v_{11}	v_{12}		v_{1j}
t_2	v_{21}	v_{22}		v_{2j}
t_3	v_{31}	v_{32}		v_{3j}
t_4	v_{41}	v_{42}		v_{4j}
t_5	v_{51}	v_{52}		v_{5j}
t_i	v_{i1}	v_{i2}		v_{ij}

図 2　ストリーム・データの例

各レコードはタイムスタンプを含む複数のカラムからなる．このような例として，コネクテッド・カーが生成する CAN データがある．CAN データは，スケーラビリティの観点からデータベースではなくファイルシステムやオブジェクト・ストレージに保存されることが一般的だが，アプリケーション開発の利便性を考慮し SQL によるアクセスをサポートすることが多い [8]．

図 2 のようなデータをストレージ・デバイスに格納する代表的なフォーマットとして，行指向形式と列指向形式がある．2.1 で述べたテープの性能特性により，行指向形式と列指向形式いずれの場合でも問題が生じる．以下にそれぞれの問題点を示す．

行指向形式

行指向形式は，図 3 のようにデータをレコード単位でストレージ・デバイスに格納する形式である．

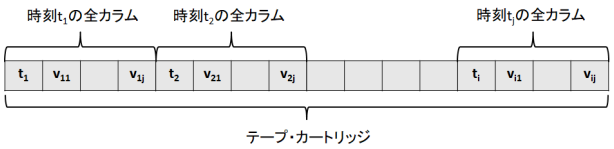

図 3　行指向形式

例えば，ある期間に生成された全レコードの全カラムを読み出す次のクエリを考える．

```
SELECT * WHERE time >= 10-01 00:00:00 AND
time <= 10-01 23:59:59
```

この場合，行指向形式は，先頭レコード (time=10-01

00:00:00) から末尾レコード (time=10-01 23:59:59) までの全レコードをシーケンシャルに読み出す．したがって，テープのシーケンシャル・アクセス性能を最大限に活用することができる．

しかし，実際にはストリーム・データはカラム間でアクセス頻度に差が生じる．例えば，次のクエリを考える．

```
SELECT C1 WHERE time >= 10-01 00:00:00 AND
time <= 10-01 23:59:59 AND C1 == V1
```

このクエリがアクセスするカラムは time と C1 の 2 つのみである．

しかし，2.1 で述べたようにラップ内のシークはテープ送りの速度で実行される．したがって，カラム数が 20 の場合，このクエリがカラム time と C1 を読み出す実効性能はテープ・ドライブのシーケンシャル・アクセス性能の 10%まで低下する．

列指向形式

列指向形式は，図 4 のようにデータをカラム単位でストレージ・デバイスに格納する形式である．

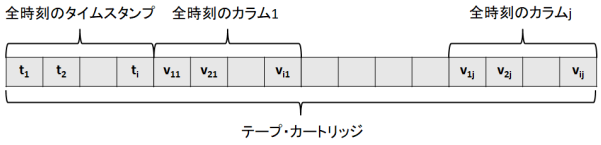

図 4　列指向形式

ここで再びカラム time とカラム C1 にアクセスするクエリを考える．この場合，カラム time のデータとカラム C1 のデータはそれぞれテープ上の連続領域に格納されている．連続領域の読み出しは高速に行うことができ，読み出す連続領域の切り替えもヘッド移動だけで済む場合は高速に行うことができる．

しかし，時系列順に到着するストリーム・データを図 4 の形式で格納するためには，テープ 1 本分のデータを別の高速なストレージ・デバイスにバッファしソートした上でテープに書き込む必要がある．これは大容量のバッファを必要とするため，コスト増大につながる．

以上のように，ストリーム・データを既存のデータ・レイアウトで格納する場合，いずれの形式でも問題が生じる．第 3 節では，クエリ実行履歴の分析結果に基づき一部のカラムを元データとは別のカートリッジにコピーし，クエリ実行時にアクセスするカートリッジを選択することで，読み出し性能を向上させる手法を提案する．

3. 提案手法

3.1 システム構成

本手法を実現するためのシステム構成を図 5 に示す．

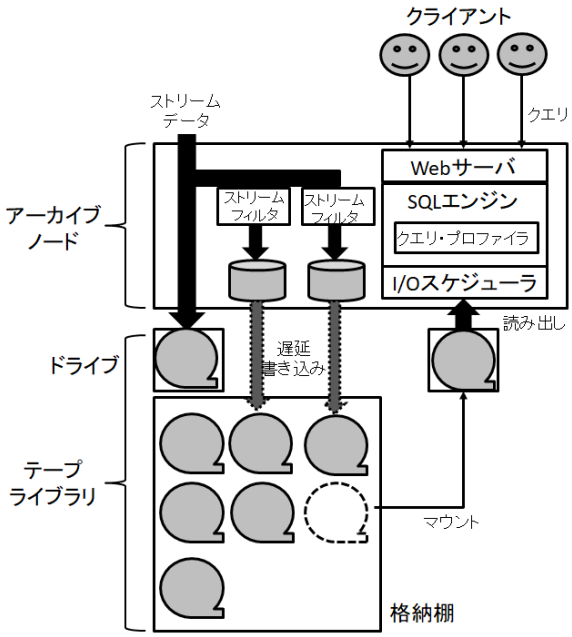

図 5　システム構成

アーカイブ・ノードはストリーム・データを受信すると，バックエンドのテープ・ライブラリに格納する．必要に応じて，少数の HDD をバッファとして使用する．テープ・ライブラリは少数のドライブと，多数のテープ・カートリッジを収容可能な保管棚を備える．

アーカイブ・ノードは，ネットワークから受信したストリーム・データをドライブ内のカートリッジに書き込む．カートリッジが一杯になると，新しいカートリッジをドライブにマウントし，排出したカートリッジは保管棚に移動する．

アーカイブ・ノードでは以下のソフトウェア・モジュールが動作する．

ストリーム・フィルタ

ストリーム・データから特定のカラム・グループを抽出し，バッファ (HDD) に一時保存する．HDD が一杯になると，ドライブのアイドル時間を見計らってカートリッジに書き出す．異なるストリーム・フィルタにより抽出されたデータは異なるカートリッジに保存される．カートリッジへの遅延書き込むを行う理由は，ストリーム・フィルタ毎に専用のドライブを用意するコストを抑えるためである．ストリーム・フィルタの追加・削除は，後述するクエリ・プロファイラの分析結果に基づき行われる．

Web サーバ

クライアントからクエリを受け付ける．

SQL エンジン

クエリを実行する．

I/O スケジューラ

クエリを実行するのに必要なテープ・カートリッジのマウントと，テープ・カートリッジに対する I/O のスケ

表 1　記号

記号	意味
Q	クエリ q の集合
C	Q がアクセスするカラム・グループ cg の集合 要素の重複は許容しないが，要素間でカラムが 部分的に重複することは許容する (例．$cg_1 = (C1, C2), cg_2 = (C1, C3)$) 全カラムを意味する特別な要素 cg_0 を常に含む
F	ストリーム・フィルタが抽出する カラム・グループ cg の集合 ($F \subset C$) C と同様 cg_0 を常に含む
$f(cg)$	cg を抽出するストリーム・フィルタ
$T(q, cg)$	$f(cg)$ が抽出したデータから q の実行に必要な データを読み出す時間 q がアクセスする全カラムが cg に含まれない 場合は $T(q, cg) = \infty$ とする
B	テープのシーケンシャル・アクセス性能
d	ストリーム・フィルタ数の上限

ジューリングを行う．

クエリ・プロファイラ

定期的に SQL エンジンから出力されたクエリ実行履歴の分析を行い，ストリーム・フィルタにより抽出すべきカラム・グループを決定する．以下，この動作間隔を単位時間（例．1 日）と呼ぶ．分析結果に基づき，ストリーム・フィルタの追加・削除を行う．

3.2　ストリーム・フィルタの追加アルゴリズム

クエリ・プロファイラは以下に述べるアルゴリズムにより，クエリ実行履歴の分析を行い追加するストリーム・フィルタを決定する．まず説明に使用する記号を表 1 に定義する．

クエリ・プロファイラの目的は次の最適化問題を解くことである．

$$\begin{aligned} &\text{minimize} \sum_{Q} \min_{cg \in F} T(q, cg) \\ &\text{subject to } |F| = d \end{aligned} \tag{1}$$

式 1 の意味は以下の通りである．元データを格納したカートリッジと，ストリーム・フィルタが抽出したデータを格納した（複数の）カートリッジがある時，クエリ q は最も高速に必要なデータを読み出すことができるカートリッジからデータを読み出すものとする．クエリ・プロファイラは，全クエリの合計アクセス時間を最小化する F を求める．

例として，以下のストリーム・データを想定する．全カラムの型は 64 ビット整数で，カラム数はタイムスタンプを含めて 20 である．カラム $time$ がタイムスタンプ（秒精度）で，カラム C_1, C_2...C_{19} がセンサ値を表す．このデータが 1 日に 1 億 5000 万レコード，サイズにして 24[GB] 生成される．ストリーム・フィルタによる抽出前のデータ（元データ）および抽出されたデータはいずれも行指向形式で格納される．

表 2　クエリ集合 Q の例

q_i	クエリ
q_1	SELECT C_1 WHERE time $\geq$ 10-02 00:00:00 AND time $\leq$ 10-02 23:59:59 AND $C_1 = V_1$
q_2	SELECT C_1,C_2 WHERE time $\geq$ 10-02 00:00:00 AND time $\leq$ 10-02 23:59:59 AND $C_1 = V_1$ AND $C_2 = V_2$
q_3	SELECT C_1,C_2 WHERE time $\geq$ 10-01 00:00:00 AND time $\leq$ 10-02 23:59:59 AND $C_1 = V_1$ AND $C_2 = V_2$
q_4	SELECT C_1,C_2,C_3 WHERE time $\geq$ 10-02 00:00:00 AND time $\leq$ 10-02 23:59:59 AND $C_1 = V_1$ AND $C_2 = V_2$ AND $C_3 = V_3$
q_5	SELECT C_4 WHERE time $\geq$ 10-02 00:00:00 AND time $\leq$ 10-02 23:59:59 AND $C_4 = V_4$
q_6	SELECT C_5 WHERE time $\geq$ 10-02 00:00:00 AND time $\leq$ 10-02 23:59:59 AND $C_5 = V_5$

表 3　Q がアクセスするカラム・グループ集合 C

cg_j	カラム・グループ
cg_0	$time, C_1, ..., C_{99}$
cg_1	$time, C_1$
cg_2	$time, C_1, C_2$
cg_3	$time, C_1, C_2, C_3$
cg_4	$time, C_4$
cg_5	$time, C_5$

毎日 0:00 にクエリ・プロファイラが前日のクエリ実行履歴の分析を行う．すなわち，単位時間は 1 日である．10-04 0:00 にクエリ・プロファイラがクエリ実行履歴の分析を行ったところ，10-03 に実行されたクエリ集合 Q は表 2 の通りであった．この時，C は表 3 のようになる．

今，$d = 1, B = 300\,(\text{MB/s})$ の条件下で式 1 を解く．手順は以下の通りである．

まず F の可能な組み合わせについて $\min_{cg \in F} T(q, cg)$ を計算する．これは，$f(cg)$ 作成後に $q \in Q$ が再び実行されると仮定した場合に，最も高速にデータを読み出せるカートリッジから読み出す時のデータ読み出し時間を意味する．

例えば，cg_2 のストリーム・フィルタを追加する場合の $\min_{cg \in F} T(q, cg)$ は次のように計算できる（$d = 1$ のため作成するストリーム・フィルタは 1 つのみ）．

q_1 の場合，$\{time, C1\} \subset \{time, C1, C2\}$ なので，元デー

タではなく抽出済みデータにアクセスすることで高速な読み出しが可能になる．ただし，クエリ実行に必要ない C_2 を読み出す（あるいは読み飛ばす）必要があるため，実効読み出し性能は $\frac{2}{3} \times B$ となる．

$$\min_{cg \in F} T(q_1, cg) = T(q_1, cg_1) = \frac{24 \times 150 \times 10^6}{300 \times 10^6} = 12$$

q_2 の場合，$\{time, C1, C2\} = \{time, C1, C2\}$ なので，元データではなく抽出済みデータにアクセスすることで高速な読み出しが可能になる．実効読み出し性能は B である．

$$\min_{cg \in F} T(q_2, cg) = T(q_2, cg_1) = \frac{24 \times 150 \times 10^6}{300 \times 10^6} = 12$$

q_3 の場合，$\{time, C1, C2\} = \{time, C1, C2\}$ なので，元データではなく抽出済みデータにアクセスすることで高速な読み出しが可能になる．実効読み出し性能は B である．

$$\min_{cg \in F} T(q_3, cg) = T(q_3, cg_1) = \frac{24 \times 300 \times 10^6}{300 \times 10^6} = 24$$

q_4 の場合，$\{time, C1, C2, C3\} \not\subset \{time, C1, C2\}$ なので，元データにアクセスしなくてはならない．全カラムを読み出す必要があるため，実効読み出し性能は $\frac{4}{20} \times B$ となる．

$$\min_{cg \in F} T(q_4, cg) = T(q_4, cg_0) = \frac{160 \times 150 \times 10^6}{300 \times 10^6} = 80$$

q_5 の場合，$\{time, C4\} \not\subset \{time, C1, C2\}$ なので，元データにアクセスしなくてはならない．全カラムを読み出す必要があるため，実効読み出し性能は $\frac{2}{20} \times B$ となる．

$$\min_{cg \in F} T(q_5, cg) = T(q_5, cg_0) = \frac{160 \times 150 \times 10^6}{300 \times 10^6} = 80$$

q_6 の場合，$\{time, C5\} \not\subset \{time, C1, C2\}$ なので，元データにアクセスしなくてはならない．全カラムを読み出す必要があるため，実効読み出し性能は $\frac{2}{20} \times B$ となる．

$$\min_{cg \in F} T(q_6, cg) = T(q_6, cg_0) = \frac{160 \times 150 \times 10^6}{300 \times 10^6} = 80$$

以上より，$F = \{cg_0, cg_2\}$ の時は，

$$\sum_{Q} \min_{cg \in F} T(q, cg) = 288$$

となる．

$F = \{cg_0, cg_1\}, \{cg_0, cg_3\}, \{cg_0, cg_4\}, \{cg_0, cg_5\}$ の時についても同様に計算した結果を表 4 に示す．参考のため，ストリーム・フィルタを追加しない場合の結果を cg_0 として示す．表 4 より，$f(cg_3)$ を追加した場合にクエリの合計アクセス時間を最小化できることがわかる．

4. 評価

4.1 実験環境

アーカイブ・ノードには PRIMERGY RX2540 M4 (Fujitsu) を使用した．Xeon Gold 6184 (Intel) を 2 ソケット (40 コア)，DDR4 の DRAM を 192(GiB) 搭載している．テープ・ライブラリとの接続には SAS3.0 対応の SCSI カードを用いた．使用した OS は CentOS 7.6 である．

テープ・ライブラリには ETERNUS LT40 S2 (Fujitsu) を使用した．LTO-7 のドライブを 2 台，カートリッジを最大 24 巻搭載可能で，LTO-7 のアクセス性能は非圧縮時で 300(MB/s) である．ドライブは 1GB のバッファを内蔵する．今回の実験にはドライブを 1 台，カートリッジを 1 巻のみ使用した．カートリッジは LTFS でフォーマットした．フォーマットのバージョンは 2.4.0 である．

表 4　F の可能な組み合わせに対する計算結果

F	$\sum_Q \min_{cg \in F} T(q, cg)$
cg_0	560
cg_1	488
cg_2	288
cg_3	240
cg_4	488
cg_5	488

4.2 ワークロード

3.2 で例として使用したデータセットに対して提案手法を適用し，全クエリの合計読み出し時間を計測した．計測に使用したクエリ集合 (Q) は表 2 と同一である．

実験手順は以下の通りである．この実験では提案手法と異なり，元データと抽出済みデータは全て LTFS ファイルとして同一カートリッジに格納した．文中の C は表 3 と同一である．

(1) 全カラム (cg_0) を含むファイルを用意する．これが元データとなる．元データは 2 日分（48[GB]）のみ用意した．
(2) $cg (cg \in C)$ を 1 つ選択し，当該 cg を元データから抽出し別ファイルに格納する．
(3) 全 $q (q \in Q)$ について，元データあるいは抽出済みデータからの読み出し時間を計測する．q がアクセスする全カラムが cg に含まれる場合は抽出済みデータを，含まれない場合は元データから読み出す．読み出し時間はファイルをオープンする時間を含まない．計測開始前にはページ・キャッシュのクリアを行い，ドライブのバッファのクリアは行わなかった．全 q について読み出し時間を計測した後，合計読み出し時間を算出した．
(4) 全 cg について，上記の手順を繰り返す．

4.3 結果

測定結果を図 6 に示す．表 4 の値を理論値として示している．参考のため，ストリーム・フィルタを作成しない場合 (cg_0) の測定も行った．

期待通り，$d = 1$ の条件下では cg_3 を抽出するストリー

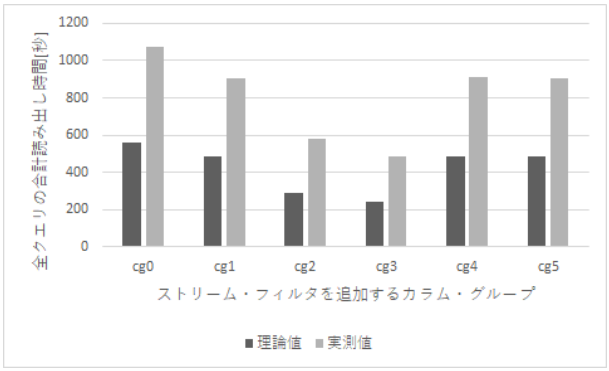

図 6 ストリーム・フィルタの有無による読み出し性能の変化

ム・フィルタを追加した場合に最も読み出し性能が良くなることがわかる．

ただし，理論値と実測値の間には常時 2 倍程度の乖離が見られる．原因究明は今後の課題とする．考えられる原因の 1 つとして，理論値の計算にシーク時間を含めていないことが挙げられる．q_3 以外は，ファイルをオープンしてから読み出す先頭レコードまでシークが必要となるため，このシーク時間が誤差の原因となりうる．

5. 関連研究

テープへのアクセスを高速化する手法については様々なものが提案されてきた．

カートリッジ交換回数の削減

テープが低コストなメディアである大きな理由の 1 つは，高価なドライブと安価なカートリッジを分離し，ドライブ数に対するカートリッジ数の比率を高めることができるためである．反面，カートリッジ交換のオーバヘッドはテープ利用時の性能低下を招く大きな要因となってきた．Bessone らは，同一カートリッジに対するクライアントからの I/O リクエストが指定した量に達するまでマウントを遅延することで，1 回のカートリッジ交換で読み出すデータの量を増やす手法を提案した [1]．Kathpal らは，テープ上のデータに対する MapReduce を高速化するため，同一カートリッジ内のデータをまとめて処理するようにタスクをスケジュールする手法を提案した [6]．Iwata らは光学メディアを対象に，データの複製間でレイアウトを変えることにより，ログ分析時のメディア交換回数を削減する手法を提案した [5]．

シーク高速化

テープ利用時のもう 1 つの大きな性能低下要因が，シークによるオーバヘッドである．この点についても，シーク高速化のための様々な手法が提案されてきた．代表的な手法が，カートリッジに対するアクセスがシーケンシャルになるように，I/O リクエストを論理アドレスが小さいものから順に並べ替えることである [6][7][15]．また，論理アドレスではなくカートリッジの物理的な構造を意識したスケジューリングも提案されている [11][13][14]．

これらの研究がスケジューリングの最適化によりテープへのアクセスを高速化しようとするのに対し，提案手法はデータ・レイアウトを含めた最適化を行うことにより，カートリッジ交換回数削減とシーク性能向上に加え，データ転送性能も向上させようとする点が異なる．

6. おわりに

本論文では，テープに保存されたストリーム・データへの SQL アクセスを高速化する手法を提案した．簡易的なベンチマークを用いた予備実験により，本手法による性能向上の可能性を評価した．今後は以下の課題に取り組む予定である．

実環境に近いワークロードでの評価

本論文では，自作の簡易的なデータセットとベンチマークを用いた評価を行った．今後は，実環境で採取したデータセットおよび統合ベンチマークを用いて，本手法の有効性を評価したい．

元データへの SQL アクセス高速化

本論文では，元データについては従来同様に行指向形式で格納するものとした．2.1 で述べた通り，テープ・カートリッジは独自の物理構造を有しており，これを利用することで元データ自体へのアクセスも高速化できる可能性がある．例えば，ラップ間のヘッド移動は比較的高速に行えることに着目し，同一カラムのデータはラップを跨いで近接する位置に格納する等の最適化が考えられる．

格納済み元データからのカラム・グループ抽出

本論文では，クエリ実行履歴の分析結果を元に，新たなストリーム・データを受信したタイミングでカラム・グループを抽出する手法を提案した．抽出処理を，データをテープ・カートリッジに書き込む前にメモリ上で行うことにより抽出のための I/O オーバヘッドを削減できるが，実際には格納済みの元データについても抽出処理を行いたいユースケースがあると思われる．そのようなニーズに対応するため，格納済みデータに対する抽出処理を効率良く行う手法についても検討したい．

正確なコスト・モデルによるカートリッジ選択

本論文では，単純化のため，クエリ実行時のカートリッジ選択はデータ転送時間のみに基づいて行った．しかし，実際にどのカートリッジから読み出すのが最速になるかは，シーク時間やカートリッジ交換のオーバヘッド等も加味して予測する必要がある．今後は，I/O スケジューラを改良し，それらの要素も考慮したスケジューリングを行えるようにしたい．

参考文献

[1] Bessone, N., Cancio, G., Murray, S. and Taurelli, G.: Increasing the efficiency of tape-based storage backends, *Journal of Physics: Conference Series*, Vol. 219, No. 1 PART 6 (online), DOI: 10.1088/1742-6596/219/6/062038 (2010).

[2] Coppola, R., Morisio, M. and Torino, P.: Connected Car : Technologies , Issues , Future Trends, *ACM Computing Surveys*, Vol. 49, No. 3, pp. 1–36 (online), DOI: http://dx.doi.org/10.1145/2971482 (2016).

[3] Eleftheriou, E., Haas, R., Jelitto, J., Lantz, M. and Pozidis, H.: Trends in storage technologies, *Data Engineering*, pp. 1–10 (2010).

[4] Fugiglando, U., Massaro, E., Santi, P., Milardo, S., Abida, K., Stahlmann, R., Netter, F. and Ratti, C.: Driving Behavior Analysis through CAN Bus Data in an Uncontrolled Environment, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 20, No. 2, pp. 737–748 (online), DOI: 10.1109/TITS.2018.2836308 (2019).

[5] Iwata, S. and Shiozawa, K.: A Simulation Result of Replicating Data with Another Layout for Reducing Media Exchange of Cold Storage, *HotStorage*, pp. 2–6 (2016).

[6] Kathpal, A. and Yasa, G. A. N.: Nakshatra: Towards running batch analytics on an archive, *MASCOTS*, Vol. 2015-Febru, No. February, pp. 479–482 (online), DOI: 10.1109/MASCOTS.2014.67 (2015).

[7] Koltsidas, I., Sarafijanovic, S., Petermann, M., Haustein, N., Seipp, H., Haas, R., Jelitto, J., Weigold, T., Childers, E., Pease, D. and Eleftheriou, E.: Seamlessly integrating disk and tape in a multi-tiered distributed file system, *ICDE*, Vol. 2015-May, pp. 1328–1339 (online), DOI: 10.1109/ICDE.2015.7113380 (2015).

[8] Luckow, A., Kennedy, K., Manhardt, F., Djerekarov, E., Vorster, B. and Apon, A.: Automotive big data: Applications, workloads and infrastructures, *BigData*, pp. 1201–1210 (online), DOI: 10.1109/BigData.2015.7363874 (2015).

[9] Lüttgau, J., Kuhn, M., Duwe, K., Alforov, Y., Betke, E., Kunkel, J. and Ludwig, T.: Survey of Storage Systems for High-Performance Computing, *Supercomputing Frontiers and Innovations*, Vol. 5, No. 1, pp. 31–58 (online), DOI: 10.14529/jsfi180103 (2018).

[10] Marosi, A. C., Lovas, R., Kisari, A. and Simonyi, E.: A novel IoT platform for the era of connected cars, *Future ioT*, Vol. 2018-Janua, pp. 1–11 (online), DOI: 10.1109/FIOT.2018.8325597 (2018).

[11] Meliá, G. C.: LTO performance, *HEPiX* (2018).

[12] Nkenyereye, L. and Jang, J.-w.: A study of Big Data solution using Hadoop to process connected Vehicle's Diagnostics data, *LNEE*, (online), DOI: 10.1007/978-981-13-1056-0 (2015).

[13] Sandst, O. and Midtstraum, R.: Improving the Access Time Performance of Serpentine Tape Drives, *ICDE* (1999).

[14] Sandsta, O. and Midtstraum, R.: Low-cost access time model for serpentine tape drives, *MSST*, pp. 116–126 (online), DOI: 10.1109/mass.1999.830005 (1999).

[15] Schaeffer, J. and Casanova, A. G.: TReqS: The tape REQuest scheduler, *Journal of Physics: Conference Series*, Vol. 331, No. PART 4 (online), DOI: 10.1088/1742-6596/331/4/042040 (2011).