関係データベースにおける処理性能の改善とチューニング作業負荷の軽減に関する検討

村上　直†

† 高エネルギー加速器研究機構（KEK） 計算科学センター
〒 305–0801 茨城県つくば市大穂 1–1
E-mail: †tadashi.murakami@kek.jp

あらまし　関係データベース管理システム（RDBMS）は，SQL による複雑なクエリのサポートやデータ整合性の保証などの優れた特性を持ち，現在でも広く使用されている．一方で，いわゆる NoSQL とよばれるデータベース管理システム（DBMS）の普及により，主に二点の問題が指摘されるようになった．一点はスキーマを事前に定義する必要性に対する煩雑さであり，もう一点は処理性能である．本研究では主に，処理性能の改善を模索する．処理性能は，近年の RDBMS では著しく改善されているほか，データウェアハウスの技術を用いればさらに向上を期待できる．本研究では関係データベースにおける処理性能の改善とチューニング作業負荷の軽減に関する検討を行い，特に分割キーを事前計算することで水平分割を支援するパーティショニング手法を 2 種類提案する．提案内容の効果を予備実験により確認し検討する．

キーワード　RDBMS，チューニング，DWH

1. はじめに

現在，いわゆる NoSQL [9] とよばれるデータベース管理システムが広く用いられている．従来の関係データベース管理システム（RDBMS）と比べて，スキーマ定義やチューニングを行うこと無しに比較的良好な性能を得ることが出来る特徴がある．また，データ整合性を犠牲にすることと引き替えにスケールアウトしやすい特徴をもつ．いっぽうで RDBMS は，SQL による複雑なクエリのサポートやデータ整合性の保証などの優れた特性を持ち，現在でも広く使用されている．RDBMS は，いわゆる NoSQL と比較して事前の知識や設計などの準備を要するシステムであるが，スキーマの事前定義が必須であり，また，チューニング無しには処理性能を得るのが難しい．本研究では主に，RDBMS の利用における処理性能の改善を模索する．

近年の RDBMS では処理性能は著しく改善されているほか，データウェアハウスの技術を用いればさらに向上を期待できる．但し，性能のチューニングには試行錯誤が必要であり，一般的には時間を多く要する．たとえばデータの水平分割（horizontal partitioning）を適切に行うと，クエリ文に対応するデータ行のストレージ上での所在を局所化でき，性能向上に大きく寄与する．しかし，水平分割のキーとする列の指定条件が複雑になると，クエリ時に適切なパーティションを絞り込むことが難しくなり，結果としてフルスキャンを実施せざるを得なくなる．

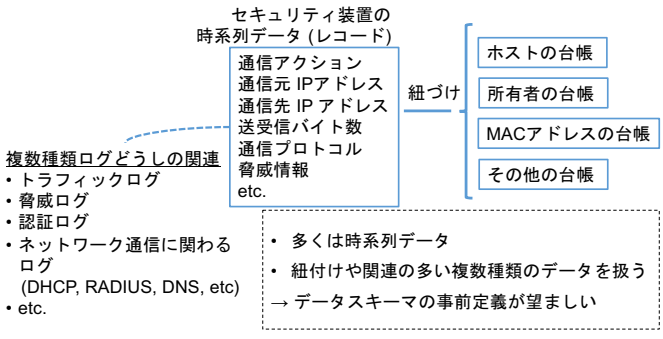

図 1　セキュリティログ管理の典型的な状況

このような状況から，セキュリティログのような，元データが大量にありながら実際に使われるデータは少なく，複雑な検索条件を使って検索対象の 1 件 1 件を特定して高速に検索できることが重要でありながら，かつ全体の傾向を掴むのも重要，といった要求に応えるのは容易ではない．

本研究では関係データベースにおける処理性能の改善とチューニング作業負荷の軽減に関する検討を行い，特に分割キーを事前計算することで水平分割を支援するパーティショニング手法を 2 種類提案する．そのうえで提案内容の効果を予備実験により確認し，比較検討する．

2. 関連研究

大容量のデータを扱う場合，いわゆるビッグデータ [8], [13] やデータウェアハウスに関係する技術は重要である．特に性能を向上させるために物理スキーマをカラムストア化（文献 [12]）する技術は重要であり，本研究ではこれを導入する．

また，データキューブの仕組みは次元数の多い大容量データを扱うために重要である．事前に次元を減じた集計値を計算してデータキューブを構成することで，集計演算を高速化できる．しかし，次元数が増えると計算すべき集計値の組み合わせが増えて計算のコストが増大するほか，データを保持するための容量を消費する．たとえば，プライバシやセキュリティなどの応用では十分に機能しない（文献 [5], [6]）．たとえば図 1 に示す

ように，通信ログの解析に RDBMS を用いる場合，少なくとも通信開始時刻，通信所要時間，通信元/通信先 IP，通信元/通信先ポート，送信/受信バイト数，通信成立有無と少なくとも 9 つの主要な次元を挙げることができ，しかも特定の通信元 IP に関係する通信を全て洗い出すといった用途が重要である．これらを適切に扱うためのデータキューブの構築には困難が伴う上に，主要な用途を果たすことが出来ない．次元数が増えた場合でもデータキューブを適用して性能向上に寄与できるような工夫は様々に行われている（文献 [10]）が，次元数が 9 にわたるデータの対処は，やはり容易ではない．

他の高速化手法として，例えば通信ログのような時系列データを扱う場合に，通信時刻をキーにしてクラスタ化する水平パーティショニング [7] の導入は，高い効果を期待できる．また，パーティション化のキーにラウンドロビンやハッシュを適用してデータベースをクラスタ化すれば，最大で台数分のリニアな性能向上を期待できる．しかしながら，列の指定条件が複雑になると，クエリ時に適切なパーティションを絞り込むことが難しくなり，結果としてフルスキャンを実施せざるを得なくなる．これを防ぐためには，パーティショニングの条件は，一般的には単一列の離散値や値範囲に基づいた単純なものとならざるを得ない．

また，データベースへのアクセス状況を計測し，主に検索や更新時のワークロードから自動的にパーティション配置を行う手法が提案されている（文献 [1], [4]）．しかしながら，これらの手法を活用するためには，データベースへのワークロードがある程度発生している必要がある．

上記から，セキュリティログのような，元データが大量にありながら実際に使われるデータは少なく，複雑な検索条件を使って検索対象の 1 件 1 件を特定して高速に検索できることが重要でありながら，かつ全体の傾向を掴むのも重要，といった要求に応えるのは容易ではない．たとえば当機構で日常的に発生している，1 日 1 億件オーダーで発生するセキュリティログを 30 日間にわたって横断的に検索し，検索対象の IP アドレスを取得するといった要求に応えることは難しい．

文献 [2] では，NoSQL に属するデータベースにデータウェアハウスの技術を援用して処理の高速化を図ろうとしている．

3. 関係データベースにおける処理性能の改善とチューニング作業負荷の軽減に関する検討

本節では，当機構で日常的に発生している 1 日 1 億件オーダーで発生するセキュリティログデータを念頭に，関係データベースにおける処理性能の改善とチューニング作業負荷の軽減に関する検討を行う．大量のデータを扱う場合，列指向データベース [12] の導入を検討できる．近年，オープンソースの列指向データベースが利用可能であり，たとえば PostgreSQL をベースとした Greenplum Database [11] や cstore_fdw [3] が知られる．カラムデータベースでは，挿入後のデータ更新には制約が課されることが多いが，検索性能が優れる，列数を増やしても検索性能が劣化しないなどの特性を備える．

処理性能の改善については，重要項目として下記を挙げることができる．

- 各フィールドに適した型の選定．データサイズの軽減や型変換処理の排除を期待できる．
- カーディナリティとデータの分布を計算．これを把握することで，様々なチューニングを実施できる．
    - カーディナリティが小さい場合は，Enum 化やスタースキーマ化を検討できる．例えばカーディナリティが 1 のカラムをまとめて 1 つのディメンジョンテーブルにすることができる．
    - カーディナリティが大きく，分布が均一の場合，パーティション化を検討できる．
- カラムの圧縮が可能な場合，分布を見て圧縮アルゴリズムを検討できる．

但し，多次元キーに対応した索引（index）や水平分割（horizontal partitioning）を構成した場合，キー配列が階層化して複雑化するため，クエリを実行した際に索引や水平分割を活かせない状況が発生すると考えられる．この状況を回避することが重要となる．

データのフォーマットは事前に指定するものとする．代表的な例として，CSV や JSON が考えられる．CSV においては，フィールド名は事前に与えておくものとする．

4. データの水平分割手法に関する提案：分割キーの事前計算に基づくパーティショニング

データの水平分割を適切に行うと，クエリ対象となる行の所在を指定されたパーティション内に局所化できるため，クエリ所要時間の短縮が期待できる．しかし，データの水平分割は標準 SQL 規格には定められておらず，RDBMS 毎に独自の実装が存在する．パーティションを指定するための方法として代表的なものは，下記の 3 種類である．

- 列の離散値に基づくパーティショニング（list partitioning）
- 列の値範囲に基づくパーティショニング（range partitioning）
- 条件分岐に基づくパーティショニング

いずれのパーティショニングにおいても，指定条件が複雑になると，クエリ時に適切なパーティションを絞り込むことが難しくなり，結果としてフルスキャンを実施せざるを得ないという問題がある．よって，一般的には単一列の離散値や値範囲に基づいた単純な条件によるパーティショニングが行われる．

本研究ではこの問題を回避するため，データ挿入時に複数列の値に基づいてパーティション決定のためのキーを事前計算し，その値に基づいて RDBMS にパーティション化を指示する手法を提案する．クエリを発行する際にこのキーを復元することでパーティション対象を特定できる．

以下，提案する手法の手順を 2 種類示す．4.1 節では列の離散値を事前計算するパーティショニング手法を示し，4.2 節では列の値範囲を事前計算するパーティショニング手法を示す．

キー算出用テーブルの例

part_id	action	src_ip_from	src_ip_to	dst_ip_from	dst_ip_to
1	deny	0.0.0.0	255.255.255.255	130.87.xx4.0	130.87.xx7.255
:					
12	deny	0.0.0.0	255.255.255.255	130.87.yy8.0	130.87.zz1.255
13	deny	0.0.0.0	255.255.255.255	0.0.0.0	255.255.255.255
14	allow	130.87.vv6.0	130.87.vv9.255	0.0.0.0	255.255.255.255
:					
22	allow	130.87.ww4.0	130.87.ww7.255	0.0.0.0	255.255.255.255
23	allow	0.0.0.0	255.255.255.255	130.87.zz4.0	130.87.zz7.255
:					
26	allow	0.0.0.0	255.255.255.255	0.0.0.0	255.255.255.255

使用例
deny かつ
dst_ip=130.87.yy9.34
を対象に検索したい

親テーブルにキー算出用テーブルをセミジョイン
・結合キー
part_id
・絞込条件
dst_ip=130.87.yy9.34 and
dst_ip between
dst_ip_from and dst_ip_to

part_id=12 で親テーブルの絞込を行える

図 2　列の離散値を事前計算するパーティショニング手法

0: action	1: is_src_first	2-8: cat_in_ip0	9-15: cat_in_ip1	16-39: ip0/24	40-63: ip1/24

(action, src_ip, dst_ip) の組を与えたときに、下記のビット計算を行って64bit の整数値を返す。返値を range partitioning のキーとして用いる。

ビット	内容
0	0: allow,　1: deny
1	0: (ip0, ip1)=(src_ip, dst_ip),　1: (ip0, ip1)=(dst_ip, src_ip)
2-8, 9-15	ip0, ip1 のカテゴリ。例えば 0:KEK, 127:Inet など
16-39, 40-63	ip0, ip1 の上位24ビット値

1 ビット目では、たとえば dst_ip が KEK のアドレスの場合にビットを立てて ip0 とip1 を反転させる策を検討できる。こうすることで、src_ip もしくは dst_ip が KEK の場合をテーブル分割の対象としてID化が可能になる。

図 3　列の値範囲を事前計算するパーティショニング手法

4.1　列の離散値を事前計算するパーティショニング手法

本手法では，対象とした複数列の値に基づいてキーを事前計算するためのテーブルを用意する．属性値がテーブルの行に示した条件に合致する場合は，対象となるパーティション ID を返す．算出されたパーティション ID による離散値に基いて，パーティショニングが実施される．このテーブルでは範囲を指定することも可能である．

図 2 に手法の例を示す．ここでは，セキュリティログ格納用テーブルの事前計算用テーブルとして，(part_id, action, src_ip_from, src_ip_to, dst_ip_from, dst_ip_to) を用意する．このテーブルに特定パーティションを算出するための条件を登録しておく．この例では，属性 action においては，指定された値を算出に用いる．属性 src_ip と dst_ip においては，各々に _from と _to を与えて範囲を指定する．行の挿入時や更新時には，用いる行の (action, src_ip, dst_ip) の組を検索条件に与えて part_id を算出し，その ID をもつパーティション化テーブルに行を格納する．行の検索時には，(action, src_ip, dst_ip) の組のうち明らかになっている列を用いて，対象テーブルにキー算出用テーブルをセミジョインする．これによって，行の挿入時や更新時に登録した part_id を検索時に特定できる．

4.2　列の値範囲を事前計算するパーティショニング手法

本手法では，対象とした複数列からパーティション化のための評価値を算出するための関数を用意する．関数が算出した値に基づいて，値範囲に基づくパーティショニングを実施する．

図 3 に手法の例を示す．この例では，(action, src_ip, dst_ip) の組を引数として与えたときに 64bit の整数値を返し，パーティション化テーブルの範囲キーとする．0 ビット目は allow/deny を仕分けする役割を果たし，1-15 ビット目でパーティションの優先選別範囲を指定する．この例では，KEK のアドレス範囲を優先する選別を行っている．

5.　予備実験と考察

5.1　予備実験の内容

高エネルギー加速器研究機構のファイアウォールから生じるアクセスログについて，検討した方法でテーブルにデータを格納し，複数パターンのクエリを発行した．対象のデータは，2015 年 12 月 1 日に発生した当機構のアクセスログであり，合計 114,036,033 件である．実験に用いたデータベースはオープンソース版 Greenplum Database 5.3.0 であり，全カラムをカラム指向ストアに設定し，zlib や RLE による列圧縮を実施した．

表 1　予備実験で作成したテーブルの概要と発行したクエリ

作成したテーブル

識別名	内容
Nix1) noidx_notype	インデックスなし、各属性の型は文字列
Nix2) noidx_noenum	インデックスなし、属性に整数や日付の型を付与
Nix3) w_types	インデックスなし、属性に整数や日付の型を付与、カーディナリティ10までの属性を enum 化
Ix) basic（基本型）	インデックスあり、属性に整数や日付の型を付与、カーディナリティ10までの属性を enum 化
以下、基本型のテーブルをパーティション化により分割	
a) t8s3d3ty3a3	時刻(8) 通信元IP(3) 通信先IP(3) タイプ(3) アクション(3)
b) t24	時刻(24)
c1) ty3a3s3d3	タイプ(3) アクション(3) 通信元IP(3) 通信先IP(3)
c2) ty2a2s2d2f2	タイプ(2) アクション(2) 通信元IP(2) 通信先IP(2) IPv4/6(2)
c3) ty2a2s5d5f2	タイプ(2) アクション(2) 通信元IP(5) 通信先IP(5) IPv4/6(2)
d) V-s3d3	通信元IP(3) 通信先IP(3)、加えてUNION ALL 文を用いたビューでアクションを水平分割
e1) L-asdL16re	アクション、通信元IP、通信先IPの組で16カテゴリ分け、検索文にレンジの範囲を付与
e2) L-asdL16	アクション、通信元IP、通信先IPの組で16カテゴリ分け
f1) R-asdR25re	アクション、通信元IP、通信先IPの組で25レンジ分け、検索文にレンジの範囲を付与
f2) R-asdR25	アクション、通信元IP、通信先IPの組で25レンジ分け
g1) T-asdT27re	アクション、通信元IP、通信先IPの組で25カテゴリ分け、検索文にレンジの範囲を付与
g2) T-asdT27	アクション、通信元IP、通信先IPの組で25カテゴリ分け

発行したクエリ

識別名	内容
1) Simple	通信元IPがある1件
2) am9-17	通信元IPがある1件かつ、通信時刻9時から17時
3) am9-17, allow	通信元IPがある1件かつ、通信時刻9時から17時かつ、allowの通信
4) Cnt-Grp	機構内外でグループ化し、グループごとに通信時刻9時から17時の通信件数を集計
5) Cnt-Grp, allow	機構内外でグループ化し、allowの通信について、グループごとに通信時刻9時から17時の通信件数を集計

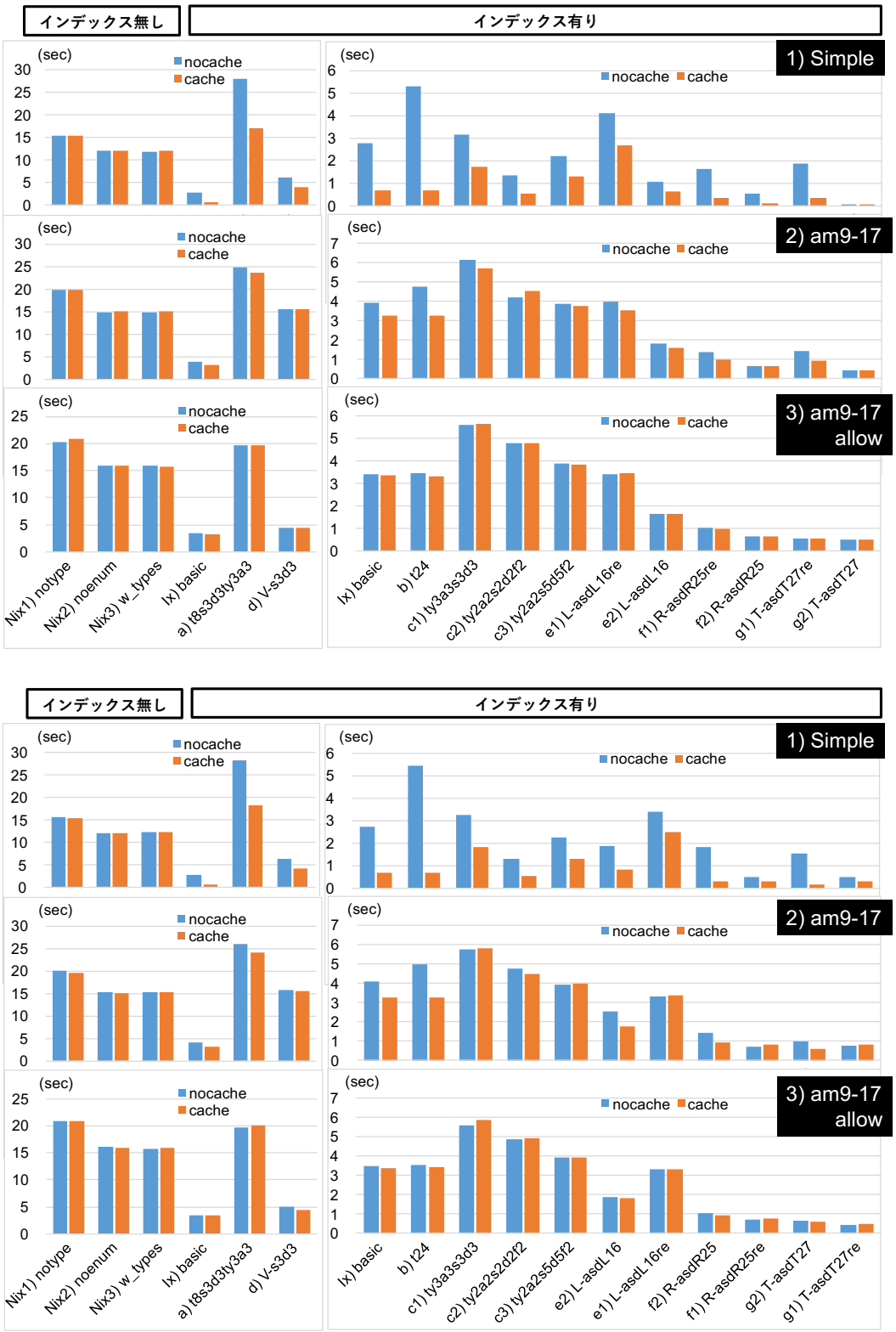

図 4　応答時間の計測結果：通信元 IP が 1 件となるクエリ．e, f, g はキーの事前計算に基づくパーティショニングを実施している．キャッシュによる影響を観察するために，e, f, g について上段と下段でクエリ発行順序を違えてある

実行したサーバのスペックは，Xeon E5-2603v3 Dual CPU，メモリ 32GB，RAID6 (SATA-HDD x 7)，Oracle Linux 7.4(RHEL 7.4 互換 OS) である．表 1 に，作成したテーブルと発行したクエリの内容を示す．Nix1) noidx_notype は，型を考慮せずに全てテキスト型としたテーブルである．これに 3. 節に示した方式を順々に適用し，テーブルのチューニングを行なった．表 1 上段の識別名 a)-g2) においては，データの分布を見ながら様々なパターンのテーブル分割を実施した．

表 1 上段に示した各テーブルに表 1 下段で示した 5 種類のクエリを発行し，応答時間を計測した結果のグラフを，図 4，5

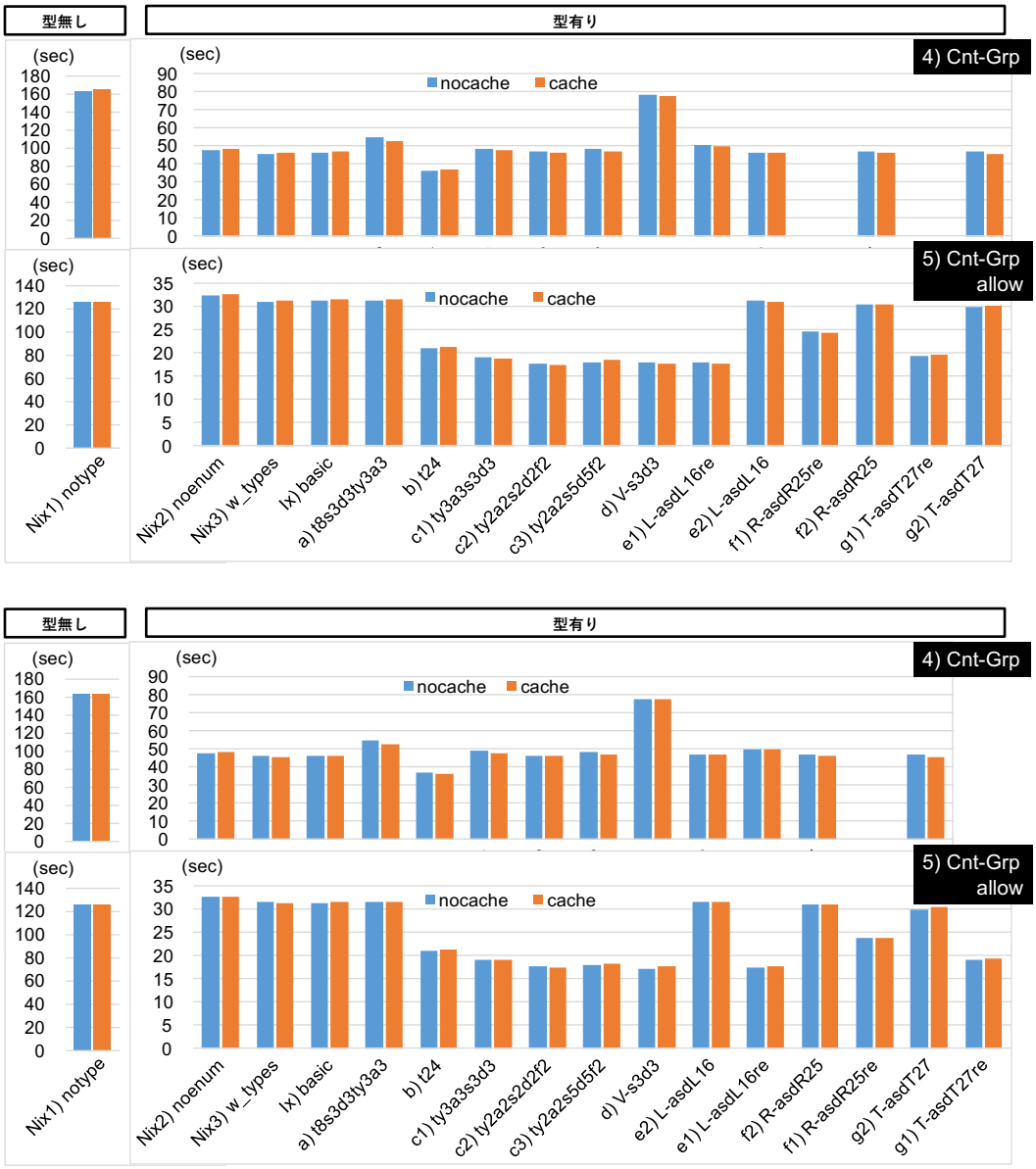

図 5　応答時間の計測結果：集計クエリ．e) f) g) はキーの事前計算に基づくパーティショニングを実施している．キャッシュによる影響を観察するために，e) f) g) について上段と下段でクエリ発行順序を違えてある

に示す．各々のグラフについて，縦軸は応答に要した時間を示し，横軸は表 1 に示した識別名に対応する．1) - 5) のクエリパターンについて，Nix1) - g2) までのテーブルに対してクエリを発行し，これを 5 回試行した．1 回目の試行を nocache として青色のグラフ，2 回目から 5 回目までの試行の平均値を cache として赤色のグラフに示した．但し nocache や cache は便宜的な分類名であり，各クエリにおいてクエリキャッシュが効いていたか否かを単純に判断は出来ない．実際の結果を見つつ個別の判断が必要である．また，e) f) g) については，検索文にレンジ範囲を付与するか否かで 2 通りのクエリを発行している．これによるキャッシュ有無の影響を調べるため，5 回試行を 2 度行い，クエリ発行順序を違えるようにした．図 4，5 について，レンジ範囲を付与したクエリを先に発行したのが上段，後に発行したのが下段である．なお比較のため，図 4 においては，識別名 Ix)basic（型とインデックスを設定してパーティションを分割しない）を左右のグラフ両方に示している．

5.2　通信元 IP が 1 件となるクエリ応答時間の計測結果と考察

本節では，図 4 に示した通信元 IP が 1 件となるクエリ応答時間の計測結果について考察する．

まず，各グラフの左右の断絶に示すように，インデックスの有無によって応答時間に大きな差が生じた．各グループの中央値で比較すると，応答時間におよそ 4 倍から 10 倍程度の差が生じている．インデックスの無い NIx1) NIx2) NIx3) での比較では，型を適切に設定することで 2 割以上の応答時間改善がみられた．

次に，インデックスの有る場合で比較する．識別名 a)t8s3d3ty3a3（時刻 8 分割，通信元 IP 3 分割，通信先 IP 3

分割，通信タイプ 3 分割，通信アクション 3 分割）の結果について，インデックスの無い場合よりも応答時間が長くなっている．識別名 a) は，考えうる様々な指標で単純にパーティション分割をおこなったものだが，得たい結果が様々なパーティションに分割されて逆効果になったと考えられる．

識別名 a)t8s3d3ty3a3 以外のインデックス有の結果については，インデックスの無い場合と比較してほとんどがおよそ 0.05 倍から 0.3 倍程度の応答時間となっており，インデックスを付与した効果を確認できた．但し，d)V-s3d3（ビューでアクションを分割し，通信元 IP 3 分割，通信先 IP 3 分割）の結果は他と比べて 2 倍以上の応答時間を要している．これは，水平分割したテーブルを UNION ALL 文を用いたビューで再結合した場合，水平分割の効果が損なわれることを示唆している．

特に，キーの事前計算に基づくパーティショニングを実施した e2)L-asdL16（通信アクション，通信元 IP，通信先 IP の組で 16 カテゴリ分け），f1,2)R-asdR25re（アクション，通信元 IP，通信先 IP の組で 25 レンジ分け），g1,2)T-asdT27（アクション，通信元 IP，通信先 IP の組で 25 カテゴリ分け）について，良好な応答時間を得ることができた．但しこの 2 つについては，検索文にパーティション名を明示的に指定する必要がある．

5.3 集計クエリ応答時間の計測結果と考察

本節では，図 5 に示した集計クエリ応答時間の計測結果について考察する．4) は通信アクション allow/deny の両方が対象であり，5) は allow のみを対象とした．いずれも，対象の時刻は 9 時から 17 時としている．

5.2 節の結果と比較した応答時間の特徴としては，インデックスの有無が結果に影響を及ぼしていないとみられることと，型の有無による影響が大きいとみられることである．これは，集計計算が対象テーブルのフルスキャンを要するためにインデックスを活用できないことと，条件取得のための型変換の要不要が結果に大きく影響したことによると考えられる．

型有りの結果比較においては，4) と 5) で結果の傾向に差異がみられた．4) においては，b)t24（時刻 24 分割）の応答結果が短く，d)V-s3d3（ビューでアクションを分割し，通信元 IP 3 分割，通信先 IP 3 分割）の応答結果が長いものとなった．b) は，時刻でテーブルを分割したことが好影響を及ぼしたと考えられる．d) の結果が芳しくなかった理由は，a)b)c) 同様（5.2 節），ビューによる分割が逆効果であったことを示唆している．5) においては，b) c1) c2) c3) d) e) の応答時間が短くなっている．b) においては時刻でテーブルを分割したこと，他は allow/deny をパーティション分割の対象の上位に加えていることが功を奏したと考えられる．

6. まとめと今後の予定

関係データベースの処理性能は，近年の RDBMS では著しく改善されているほか，データウェアハウスの技術を用いればさらに向上を期待できる．本稿では，関係データベースにおける処理性能の改善とチューニング作業負荷の軽減に関する検討を行った．カラム指向の関係データベースを対象として，考えられる様々なチューニングパターンを試行し，複数パターンのクエリについて応答時間を計測した．その結果，検索対象が少ないクエリにおいてはインデックスが，検索対象が多いクエリにおいてはカラムの型を適切に指定することが応答時間に大きな影響を及ぼすことを，実際に確認できた．また，テーブルの水平分割においては，分割の軸によって応答時間が大きく変化するほか，対象のクエリと親和性の悪い分割軸を用いた場合には，性能が却って悪化することが確認できた．

このような状況において，提案手法である列の離散値や値範囲を事前計算するパーティショニング手法が様々なクエリに対して有効に作用した．しかし，その理由については更に調査が必要であると考えている．そして，様々なクエリパターンに対して良好なパフォーマンスを得られるようにするための試行作業を軽減できる仕組みを模索する予定である．また，いわゆる NoSQL のデータベースや従来の行指向関係データベースに対して同様のチューニングを行ってクエリを発行した際の応答速度についても比較検討したいと考えている．

文　献

[1] S. Agrawal, V. Narasayya, and B. Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 359–370, New York, NY, USA, 2004. ACM.

[2] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, and R. Tournier. Implementing multidimensional data warehouses into nosql. In *Proceedings of the 17th International Conference on Enterprise Information Systems*, pages 172–183, 2015.

[3] Citus Data, Inc. cstore_fdw. `https://citusdata.github.io/cstore_fdw/`, 2017.

[4] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57, Sept. 2010.

[5] A. Cuzzocrea. Privacy and security of big data: Current challenges and future research perspectives. In *Proceedings of the First International Workshop on Privacy and Secuirty of Big Data*, PSBD '14, pages 45–47, New York, NY, USA, 2014. ACM.

[6] A. Cuzzocrea and R. Moussa. Multidimensional database modeling: Literature survey and research agenda in the big data era. In *2017 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6, May 2017.

[7] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[8] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[9] C. Mohan. History repeats itself: sensible and nonsensql aspects of the nosql hoopla. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 11–16. ACM, 2013.

[10] K. Morfonios, S. Konakas, Y. Ioannidis, and N. Kotsis. Rolap implementations of the data cube. *ACM Comput. Surv.*, 39(4), Nov. 2007.

[11] Pivotal Software, Inc. Download Greenplum Database. `http://greenplum.org/download/`, 2017.

[12] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[13] A. R. Zope, A. Vidhate, and N. Harale. Data minding approach in security information and event management. *J. Future Comput. Commun*, 2013.