

光学赤外線天文観測データアーカイブシステムにおける 検索高速化の研究2

小澤武揚, 中島 康, 小野里宏樹, 内山久和

(2023 年 12 月 26 日受付; 2024 年 3 月 7 日受理)

A Study to Speed Up Searches in the Optical-infrared Astronomical Observation Data Archive System 2

Takeaki OZAWA, Yasushi NAKAJIMA, Hiroki ONOZATO, Hisakazu UCHIYAMA

Abstract

In this paper, we report on experiments conducted to speed up searches in the optical-infrared astronomical observation data archive system and their results. High-speed or wide-field imaging instruments like Tomo-e Gozen and Hyper Suprime-Cam generate a huge amount of observational data. Such recent surge in data volume leads to a slowdown in data search speed in the data archive system. We conduct experiments to speed up searches using the database of SMOKA/Tomo-e Gozen system, which archives Tomo-e Gozen data, and its SQL queries. Concretely, we explore optimal table partitioning and parallel query settings, SSD-based table file placement impact, and PostGIS R-tree spatial indexing effects. The results reveal that increasing the number of table partitions and utilizing SSDs reduce cache-invalidated execution time and using PostGIS shortens cache-enabled spatial query execution time. The maximum reduce rates of each experiment are about 83 %, 98 %, and 91 %, respectively. This paper elaborates on the details of these experiments and discusses the suitable database settings for the data archive systems having a huge amount of observation data.

概要

本論文では、光学赤外線天文観測データアーカイブシステムの検索高速化のためにおこなった実験とその結果について報告する。高速撮像観測装置である Tomo-e Gozen や広視野撮像観測装置である Hyper Suprime-Cam では、日々膨大な数の観測データが生成されている。これらの観測データを公開するデータアーカイブシステムでは、データ数の膨大化により検索速度が低下する。我々は、Tomo-e Gozen のデータをアーカイブする SMOKA/Tomo-e Gozen システムのデータベースとその検索用 SQL クエリを使って、検索速度の高速化実験をおこなった。具体的には、最適なテーブル分割数とパラレルクエリの設定、SSD 上へのテーブルファイルの配置、並びに PostGIS R-tree 空間インデックスの効果を検討した。第一と第二の実験ではキャッシュ無効時の実行時間を、第三の実験ではキャッシュ有効時の領域検索の実行時間を短縮できることがわかった。それぞれの実験の実行時間の減少率は最大で 83 %, 98 %, 91 % 程度であった。本論文ではこれらの実験の詳細を述べると共に、膨大な数の観測データを有するデータアーカイブシステムに最適なデータベースの設定等について論ずる。



この記事はクリエイティブ・コモンズ[表示 4.0 国際]ライセンスの下に提供されています。
<https://creativecommons.org/licenses/by/4.0/deed.ja>

1 はじめに

天文観測データ（以下、観測データ）はある時刻・ある天域の状態を捉えた再現不可能な唯一無二のデータである。これらを長期間保管し、研究教育活動のため世界に公開する天文観測データアーカイブシステムは、天文学における基盤設備と言える。国立天文台天文データセンターでは、SMOKAシステム[1, 2, 3, 4, 5, 6, 7]やその派生システムであるSMOKA/Tomo-e Gozenシステム[8]を運用し、観測データの保管と公開をおこなっている。

観測データの公開に当たっては、観測データのメタ情報と管理情報の双方を管理する必要がある。前者はFITSファイルのヘッダに記述された天体の座標や観測日時等の情報である。後者は計算機システム内でのFITSファイルの在処やそのハッシュ値、データの公開日や公開可否判定等の情報である。天文データセンターではデータベース管理システム（以下、DBMS）であるPostgreSQL [9]を使って双方の情報を管理している。

光学赤外線天文学の観測装置の中には、膨大な数の観測データを生成するものがある。例えば、東京大学木曽観測所105 cm シュミット望遠鏡に搭載された高速撮像観測装置であるTomo-e Gozen [10]は、一月あたり平均150万フレームのスタック済みデータを生成する[8]。すばる望遠鏡に搭載された広視野撮像観測装置であるHyper Suprime-Cam [11, 12, 13, 14]は、一月あたり平均20万フレームのデータを生成する¹⁾。観測データ数の膨大化は、テーブルファイルとインデックスファイルを肥大化させ、これらを計算機のメモリにキャッシュしきれない状況を招く。そのような状況下では検索時にディスクアクセスが発生し、検索速度が大きく低下する可能性がある。

ディスクアクセス発生時の検索速度を高速化させる方策として、SQLクエリ実行時にディスクから読み込まれるデータ量の低減、SQLクエリの処理速度の向上、ディスク読み込み速度自体の向上等が考えられる。我々はテーブルの分割[15]、パラレルクエリの有効化[16]、Solid State Drive (SSD) を利用したデータI/Oの高速化によってこれらを実現できると考え、前論文[17]でSMOKA/Tomo-e Gozenシステムのデータベースを使った検索速度の高速化実験をおこなった。

SMOKA/Tomo-e Gozenシステムのデータベースは、観測データのメタ情報用テーブルと管理情報用テーブル、並びに両テーブルを結合したビューから構成される。赤道座標から目的の観測データをみつけ出すピン

ポイント検索とラフ検索、観測日から観測データをみつけ出すカレンダー検索のSQLクエリが実装されており、それぞれビューを問い合わせ先としている。

前論文では約2800万フレームの観測データを同データベースに入力し、実験をおこなった。ディスクアクセスが発生する状況を再現するため、OSのページキャッシュとPostgreSQLの共有キャッシュを削除した状態でSQLクエリを実行した。テーブル分割とパラレルクエリを併用した実験では、メタ情報用テーブルを赤経で分割し、パラレルクエリを有効化することで、ビューに対するピンポイント検索とラフ検索を高速化できることがわかった。SSDを利用した実験では、インデックスファイルをSSD上に配置することで、検索速度を高速化できる場合があることがわかった。一方、検索速度の高速化に寄与するであろう、最適なテーブル分割数とパラレルクエリの設定値の調査、SSD上へテーブルファイルを配置した場合の検索速度の調査、PostGIS [18]等を利用したR-tree空間インデックス[19]の検証等が、課題として残っていた。

本論文では、テーブル分割とパラレルクエリを併用した際の最適なテーブル分割数とパラレルクエリの設定値の調査（テーブル分割+パラレルクエリ実験）、テーブルファイルとインデックスファイルをSSD上に配置した場合の検索速度の調査（SSD実験）、PostGISによるR-tree空間インデックスを利用した場合の検索速度の調査（PostGIS実験）をおこない、膨大な観測データ数を有する光学赤外線天文観測データアーカイブシステムのデータベースの検索速度の高速化のためのデータベースの設定等について論ずる。

本論文の2節では実験に利用した技術について紹介する。3節では実験に使用した計算機やデータベース等の実験環境について紹介する。4節から6節ではテーブル分割+パラレルクエリ実験、SSD実験、並びにPostGIS実験、それぞれの実験結果と考察について述べる。7節では本論文のまとめと今後の課題について論ずる。

2 実験に利用した技術

本論文ではテーブル分割+パラレルクエリ実験、SSD実験、PostGIS実験という3種類の実験をおこなった。以下ではそれぞれの実験で利用した技術について紹介する。

テーブル分割+パラレルクエリ実験

テーブル分割とは、複数の小さなテーブル（子テ

1) 2023年9月現在。SMOKAシステムのデータベースより算出。

ブル)を一つの大きなテーブル(親テーブル)として扱う技術である[15]. 本実験で使用する PostgreSQL 12.7では、宣言的パーティショニングと呼ばれるテーブル分割用構文が用意されている. 分割方法として範囲分割, リスト分割, ハッシュ分割を選択できる[15]. 本実験では範囲分割を使って赤経方向にテーブルを分割する. 範囲分割は任意の列(分割キーと呼ばれる)が持つ値を任意に設定した範囲毎に分け, 各範囲毎に作成した子テーブルにデータを分配する方法である. 分割キーが問い合わせの検索条件に含まれる場合, 検索条件に一致する子テーブルのみが処理対象となることで検索速度が高速化される[20]. 読み込みページ数[21]が減少することでも検索速度が高速化されるものと考えられる(前論文[17] 4.1節を参照).

パラレルクエリとは, 複数のCPUを活用する実行計画を生成する技術である[16]. PostgreSQL 12.7ではデフォルトで有効であり, テーブルのスキャン処理, テーブルの結合処理, 検索結果の結合処理等を並列化できる[22].

SSD実験

SSDは半導体素子へ電氣的にデータの読み書きをおこなうため, 磁氣的に読み書きをおこなう Hard Disk Drive (HDD) に比べデータの読み書き速度が極めて速い[23]. 特にランダムアクセスが発生するデータベー

スのようなアプリケーションでその優位性が顕著に現れる[24]. 本論文の実験では予算の都合から通信速度が6 GbpsのSATA SSDを使用した. なお数十Gbpsの転送速度を有するNVMe SSD等のSSDも存在する.

PostGIS実験

PostGIS [18]はPostgreSQLの拡張機能であり, GIS(地理情報システム)オブジェクト(球面上の点や線分など)を格納することができる. R-tree空間インデックス[19]をサポートし, GISオブジェクトの処理を行うための関数などの機能を併せ持つ. 天文観測データアーカイブシステムの主な検索対象キーである天球座標は二次元座標であるため, その検索においてB-treeインデックス[25]よりもR-tree空間インデックスを利用することでより高速な検索が期待される. 本論文の実験ではPostGIS 3.3を使用した.

3 実験環境

本論文と前論文[17]の実験結果を比較可能にするため, 基本的に前論文と同じ実験用計算機, 実験用DBMS, 実験用データ, 実験用データベース, 実験用SQLクエリ, 実行時間測定方法を使用した.

表1 実験用計算機諸元.

実験用サーバ	CPU	Intel Xeon Silver 4214R 2.4 GHz 12 core × 2
	RAM	DDR4 2400 RDIMM 32 GB × 4
	OS	CentOS 7.9
テーブル空間用ディスク	HDD	3 TB 3.5 inch SATA 6.0 Gbps 7200 rpm
	SSD	960 GB 2.5 inch SATA 6.0 Gbps

表2 実験用計算機のpostgresql.confの設定.

パラメータ	設定値
shared_buffers	16 GB
work_mem	16772 kB
maintenance_work_mem	2 GB
max_parallel_workers_per_gather	0
wal_buffers	16 MB
max_wal_size	30 GB
min_wal_size	1 GB
checkpoint_timeout	30 min
checkpoint_completion_target	0.7

実験用計算機

実験に使用した計算機は、前論文で使用したものと同一個体である（表1）。前論文では同計算機にテーブル空間用の7本の8TB HDD（3.5 inch SATA 6.0 Gbps 7200 rpm）を搭載して実験をおこなっていたが、SMOKA システムの運用上の都合から、これら全てを3TB HDD（3.5 inch SATA 6.0 Gbps 7200 rpm）に換装して本論文の実験をおこなった。本論文ではそれぞれの3TB HDDを、hdd1, hdd2, hdd3, hdd4, hdd5, hdd6, hdd7と呼称する。なおSSD実験ではhdd7をSSD（960 GB 2.5 inch SATA 6.0 Gbps）に換装して実験をおこなった。

実験用DBMS

DBMSはSMOKA/Tomo-e Gozenシステムと同じPostgreSQL 12.7を使用した。postgresql.confの設定もSMOKA/Tomo-e Gozenシステムと同じである（表2）。

実験用データ

実験に使用したデータは前論文と同じく、Tomo-e Gozenの全天サーベイ、高頻度サーベイ、超新星サーベイの3プロジェクト^{2) 3)}で得られた観測データに、一

次処理（バイアス、ダーク、フラット補正）と位置校正並びにスタック処理を施したスタック済みデータのうち、観測日が2019-10-01から2021-05-14までの計27,918,958フレームのデータである（前論文の3.1節実験用データを参照）。

実験用データベース

実験用データベースは前論文と同じく、SMOKA/Tomo-e Gozenシステムのデータベースと同等の構造を持つ。

SMOKA/Tomo-e Gozenシステムのデータベースは二つのテーブルと一つのビューから構成される（図1）。tmq_kisoテーブルはスタック済みデータのメタ情報が格納されるテーブルである。filemng_tmqテーブルはスタック済みデータの管理情報が格納されるテーブルである。両テーブルを結合したものがtmq_smokaビューである。詳細については前論文2節を参照されたい。

- 2) http://www.ioa.s.u-tokyo.ac.jp/kisohp/RESEARCH/symp2021/kisosymp2021_Sako.pdf
 3) http://www.ioa.s.u-tokyo.ac.jp/kisohp/RESEARCH/symp2021/kisosymp2021_Tominaga.pdf

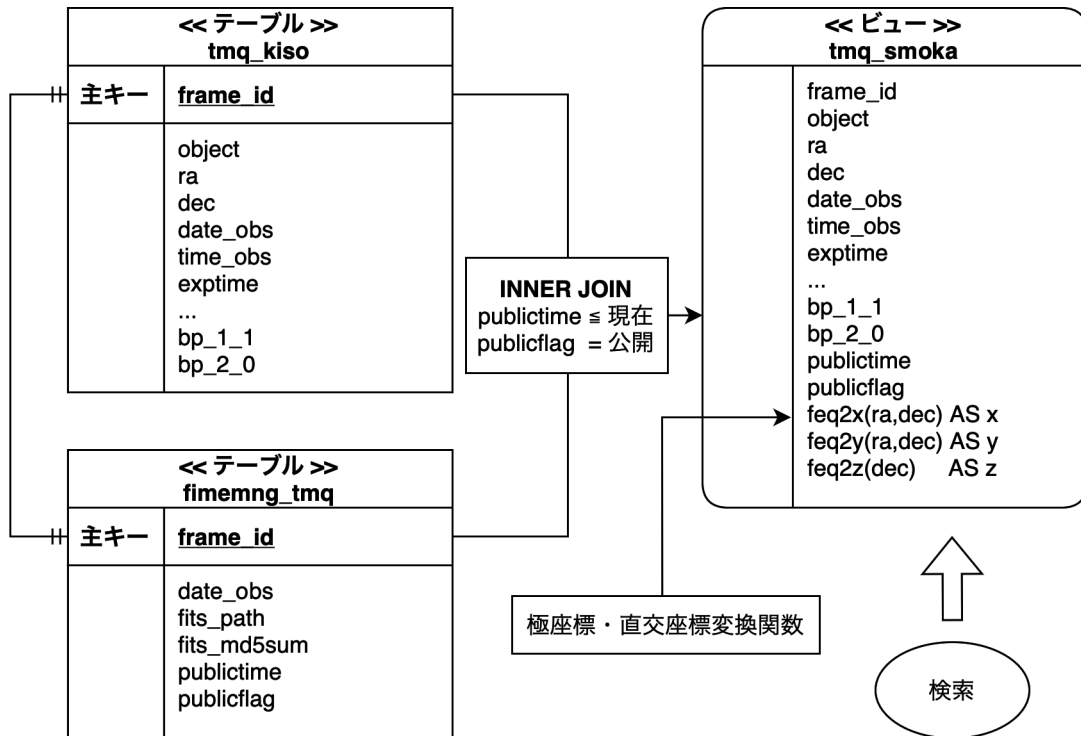


図1 SMOKA/Tomo-e Gozenシステムのデータベースの概略。スタック済みデータのメタ情報と管理情報は、frame_idを主キーとしてそれぞれtmq_kisoテーブルとfilemng_tmqテーブルに格納される。検索条件となる列にB-treeインデックスをそれぞれ設定している。両テーブルをframe_idを結合条件としてINNER JOINしたものがtmq_smokaビューである。publictimeが現在よりも古くかつpublicflagが公開となっているデータのみを検索対象とする検索条件を課している。またラフ検索のため、極座標・直交座標変換関数を加えている。問い合わせはtmq_smokaビューに対して実行される。

実験用データベースも同様に、二つのテーブルと一つのビューから構成される。tmq_kiso テーブル、filemng_tmq テーブル、tmq_smoka ビューの代わりに、それぞれの細部を変更した tbl_fheader* テーブル、tbl_filemng* テーブル、view_tomoe* ビューから構成される（‘*’ は0文字以上の文字列を表すものとする）。約2800万フレームのスタック済みデータのメタ情報と管理情報を、tbl_fheader* と tbl_filemng* にそれぞれ入力した。全てのスタック済みデータが検索対象となるよう、view_tomoe* の publictime に対する検索条件を無効化した。各テーブルとビューの詳細については各実験の節で紹介する。

実験用 SQL クエリ

実験用 SQL クエリは前論文と同じく、SMOKA/Tomo-e Gozen システムのピンポイント検索、ラフ検索、カレンダー検索の SQL クエリと同等の定義を持つ。

ピンポイント検索は任意の赤経赤緯の座標点（以降、検索座標点）が視野に含まれるフレームを検索するもの、ラフ検索は任意の検索座標点を望遠鏡の視野範囲内におおむね含むフレームを検索するもの、カレンダー検索は任意の年月日に観測されたフレームを検索するものである。各検索の詳細については SMOKA/Tomo-e Gozen システムの開発論文[8]並びに前論文2節を参照されたい。

実験用 SQL クエリでは、SMOKA/Tomo-e Gozen システムの各 SQL クエリの問い合わせ先（FROM 句の値）を tbl_fheader* または view_tomoe* に置き換えた。また前論文と同様に、ピンポイント検索とラフ検索では検索座標点を、カレンダー検索では検索する年月日を固定した（前論文3.2節参照）。

実行時間測定方法

実行時間の測定は前論文と同じく、実験用 SQL クエリの先頭に EXPLAIN ANALYZE 句[26]を追加することでおこなった。EXPLAIN ANALYZE 句が出力する実行計画中の Execution time の値（単位：ms）を、当該 SQL クエリの実行時間とした。また EXPLAIN (ANALYZE, BUFFERS) 句を実行し、その実行計画からキャッシュ無効時における読み出しページ数[21]を調査した。

ディスクからのページ読み込みが発生する場合としない場合双方の検索速度を調べるため、キャッシュ無効時と有効時の実行時間を測定した。キャッシュ無効時は OS のページキャッシュと PostgreSQL の共有キャッシュを削除した状態、キャッシュ無効時はそれらを削除していない状態とする。計算機を再起動することでキャッシュの削除をおこない、再起動直後に実施した問い合わせをキャッシュ無効時の測定、そ

れ以外の問い合わせをキャッシュ有効時の測定とした。キャッシュ無効時並びに有効時とも実行時間を3回測定し、その平均値と標準誤差を求めた。

4 テーブル分割+パラレルクエリ実験

SMOKA/Tomo-e Gozen システムのデータベースでは、前論文[17]の研究結果に基づき、2023年1月よりテーブル分割とパラレルクエリを併用して検索速度の高速化を図っている。テーブル分割は crval1 列（赤経）を分割キーとして、tmq_kiso テーブルを赤経10度毎に36分割している。パラレルクエリは PostgreSQL のパラレルクエリに関するパラメータである、max_parallel_workers_per_gather, max_worker_processes, max_parallel_workers [27] の設定値（以下、パラレルクエリのパラメータ設定値）を ‘8’ に設定し、ユーザ定義関数を ‘PARALLEL SAFE’ [28] へ変更している。

本実験では検索速度のさらなる高速化のため、最適なテーブル分割数とパラレルクエリのパラメータ設定値の組み合わせについて調査する。

4.1 実験方法

図2は実験条件の組み合わせを示したものである。使用する実験用 SQL クエリはピンポイント検索、ラフ検索、カレンダー検索の3種類である。これらの実験用 SQL クエリを9種類の実験用ビュー（view_tomoe*）に対して実行する。それぞれの実験用ビューは crval1 列を分割キーとして0, 6, 12, 24, 36, 48, 90, 180, 360分割されたメタ情報テーブル（tbl_fheader*）と管理情報テーブル（tbl_filemng）から構成される。パラレルクエリのパラメータ設定値として8, 16, 32, 64の4通りを試す。SQL クエリ実行時のキャッシュの状態として、計算機の再起動によりキャッシュの削除をおこなったキャッシュ無効の状態と、キャッシュ有効の状態の2通りを試す。これらの216通りの組み合わせについてそれぞれ実行時間を3回測定し、その平均値と標準誤差を求めた。なお前論文のパラレルクエリの実験ではテーブルに対する検索速度の高速化が認められなかったため、本実験ではビューのみを調査対象とした。

表3は作成したテーブルとビューの一覧である。メタ情報テーブルとして非分割の tbl_fheader と、crval1 列を分割キーとして同テーブルを6, 12, 24, 36, 48, 90, 180, 360分割した tbl_fheader_raXXX を作成した。これらのテーブルと管理情報テーブルである tbl_filemng から実験用ビューである view_tomoe* をそれぞれ作成した。テーブルとビューの作成手順は次のとおりである。

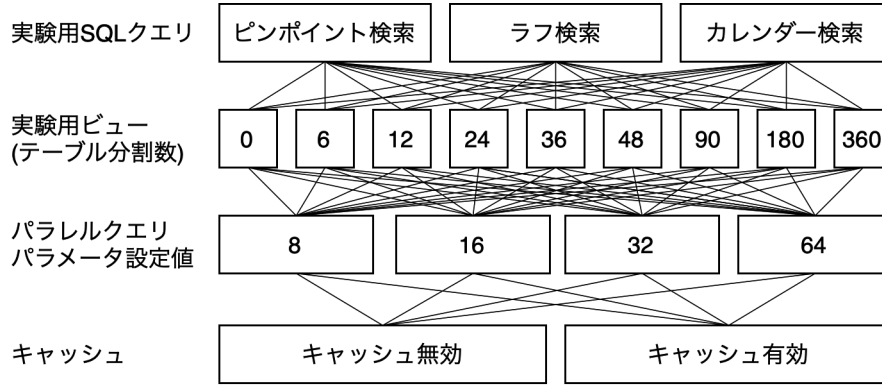


図2 テーブル分割+パラレルクエリ実験の実験条件の組み合わせ。実験用SQLクエリが3種類、テーブル分割数が9通り、パラレルクエリのパラメータ設定値が4通り、キャッシュの状態が2通りある。これら216通りの組み合わせについてそれぞれ実行時間を測定した。

表3 作成したテーブルとビューの一覧。

メタ情報テーブル			管理情報テーブル	実験用ビュー
テーブル名	分割数	分割範囲	テーブル名	ビュー名
tbl_fheader	0	N/A	tbl_filemng	view_tomoe
tbl_fheader_ra006	6	赤経 60 度毎	tbl_filemng	view_tomoe_ra006
tbl_fheader_ra012	12	赤経 30 度毎	tbl_filemng	view_tomoe_ra012
tbl_fheader_ra024	24	赤経 15 度毎	tbl_filemng	view_tomoe_ra024
tbl_fheader_ra036	36	赤経 10 度毎	tbl_filemng	view_tomoe_ra036
tbl_fheader_ra048	48	赤経 7.5 度毎	tbl_filemng	view_tomoe_ra048
tbl_fheader_ra090	90	赤経 4 度毎	tbl_filemng	view_tomoe_ra090
tbl_fheader_ra180	180	赤経 2 度毎	tbl_filemng	view_tomoe_ra180
tbl_fheader_ra360	360	赤経 1 度毎	tbl_filemng	view_tomoe_ra360

はじめに SMOKA/Tomo-e Gozen システムの `tmq_kiso` テーブル、`filemng_tmq` テーブル、`tmq_kiso` ビューと同じテーブル定義とインデックスを持つ `tbl_fheader`、`tbl_filemng`、`view_tomoe` を作成した（テーブル定義の詳細は前論文補遺 A の SQL 4, SQL 5, SQL 6 を参照）。`tbl_fheader` と `tbl_filemng` には、約 2800 万フレームのスタック済みデータのメタ情報と管理情報をそれぞれ入力した。次に PostgreSQL の宣言的パーティショニングによる範囲分割を使って、`tbl_fheader_raXXX` を作成した。子テーブルは赤経 0 度を基点に、表 3 の分割範囲に示した赤経毎に作成した。例えば `tbl_fheader_ra006` は赤経 60 度毎に区切られた ($0^\circ \leq \text{crval1} < 60^\circ$, $60^\circ \leq \text{crval1} < 120^\circ$, ..., $300^\circ \leq \text{crval1} < 360^\circ$)、6 個の子テーブルを持つ。データは `tbl_fheader` から `frame_id` 順に INSERT コマンド [29] を使って入力した。最後に `tbl_fheader_raXXX` と `tbl_filemng` から、`view_tomoe*` を作成した。

テーブルとインデックスの作成時にテーブル空間を指定することで、テーブルファイルとインデックス

ファイルを `hdd1` から `hdd7` までの任意のディスクに配置した。`tbl_fheader` と `tbl_filemng` のテーブルファイルとインデックスファイルは `hdd1` 上に配置した。これは単一のディスクでデータベースが運用されることを想定したものである。`tbl_fheader_raXXX` の親テーブルのテーブルファイルとインデックスファイルは `hdd2` に配置した。子テーブルのテーブルファイルは、隣り合った区間の子テーブルが異なるディスク上に来るように `hdd2` から `hdd7` に配置した（前論文表 13 の `tbl_fheader_ra_6d` を参照）。これは検索領域が複数の子テーブルにまたがった場合に、ディスク I/O の低下による検索速度の低下を避けるためである。

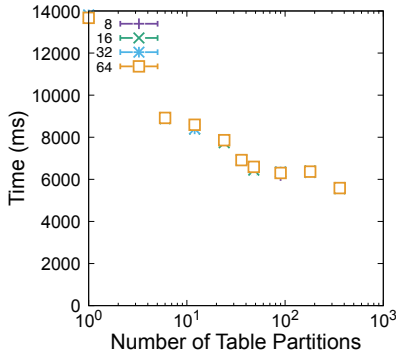
パラレルクエリのパラメータ設定値は 8, 16, 32, 64 の 4 通りを試した。本実験では三つのパラメータ (`max_parallel_workers_per_gather`, `max_worker_processes`, `max_parallel_workers`) の設定値を変更するが、三つとも同じ値に設定した。

4.2 実験結果

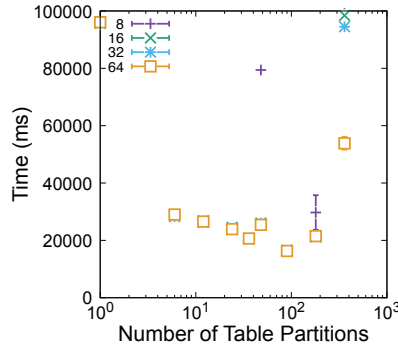
図3は、view_tomoe*に対して実験用SQLクエリを実行した際の、実行時間の平均値（Time）に対するテーブル分割数（Number of Table Partitions）のグラフである。実行時間の単位はms、誤差棒は実行時間の標準誤差を表す。それぞれのグラフでパラレルクエリのパラメータ設定値が8, 16, 32, 64である時の実行時間をプロットしている。ただし、あるテーブル分割数において各パラメータ設定値の実行時間が近接している場合はデータ点が重なるため、重なったデータ点中でパラメータ設定値が最も大きいものしか表示されていない。また図3(a)では、パラメータ設定値が8の場合の48分割時のピンポイント検索の実行時間（30,451.5 ms）がグラフ圏外となっている。キャッシュ無効時、ピンポイント検索とラフ検索ではテーブル分割数が増えるにつれ実行時間が減少した。ただし、ラフ検索では180分割以上で実行時間が増加していた。カレンダー検索ではテーブル分割数が増えるにつれ実行時間が増加した。パラレルクエリのパラメータ設定値の違いによる実行時間の系統的变化はみられなかった。

キャッシュ有効時、ピンポイント検索ではテーブル分割数が24分割までは実行時間が変わらないものの、36分割以上では分割数が増えるにつれ実行時間が減少する傾向がみられた。ラフ検索ではテーブル分割数の増加と実行時間との間に明確な関係性がみられなかった。カレンダー検索では90分割までは実行時間が減少していくが、180分割以上では増加した。パラレルクエリのパラメータ設定値の違いによる実行時間の系統的变化はみられなかった。表4は図3の作成に用いた実行時間の数値データである。

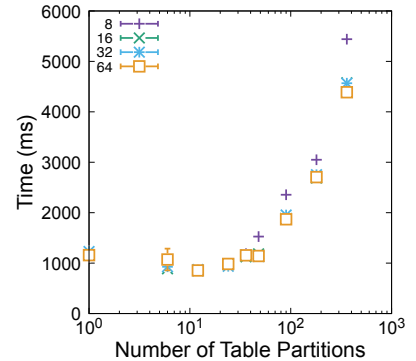
表5は、view_tomoe*に対して実験用SQLクエリを実行した際の、キャッシュ無効時における実行計画から各種情報を抽出したものである。ワーカ起動数はGatherまたはGather Merge処理のWorkers Launchedの値である。被スキャン子テーブル数はスキャンが実行されたtbl_fheader_raXXXの子テーブルの数である。子テーブルスキャン方式はtbl_fheader_raXXXの子テーブルに対して実行されたスキャンの方式であり、I, PI, PBはそれぞれIndex Scan, Parallel Index Scan, Parallel Bitmap Heap Scanを表す。子テーブルスキャン時間はtbl_fheader_raXXXの子テーブルのスキャンが



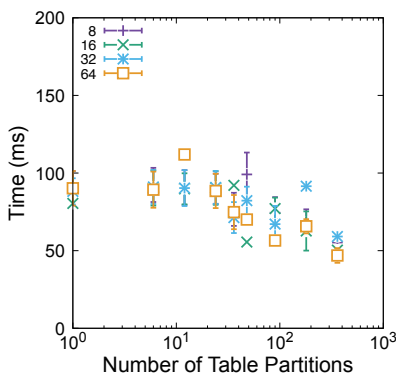
(a) ピンポイント検索（キャッシュ無効）



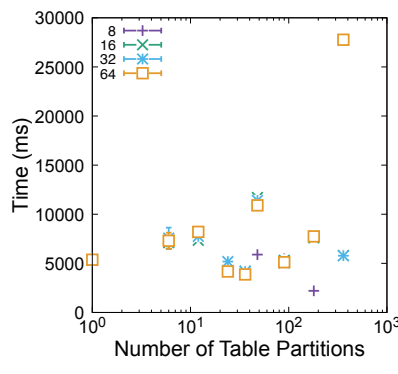
(b) ラフ検索（キャッシュ無効）



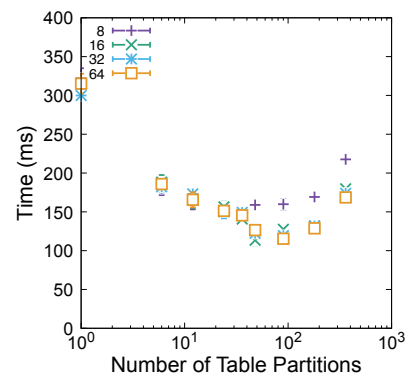
(c) カレンダー検索（キャッシュ無効）



(d) ピンポイント検索（キャッシュ有効）



(e) ラフ検索（キャッシュ有効）



(f) カレンダー検索（キャッシュ有効）

図3 view_tomoe_raXXXに対して実験用SQLクエリを実行した際の、キャッシュ無効時並びに有効時における実行時間の平均値に対するテーブル分割数。実行時間の単位はms、誤差棒は実行時間の標準誤差、凡例はパラレルクエリのパラメータ設定値。

完了した時点での経過時間である。経過時間とは、実行計画中の各処理の **actual time** に表示される二つの時間のうちの右側の時間のことを指す。スキャンされる子テーブルが1個の場合はその子テーブルのスキャンが完了した時点での経過時間を、スキャンされる子テーブルが複数ある場合は **Parallel Append** 処理が完了した時点での経過時間を記載している。なお、ワーカー起動数、被スキャン子テーブル数、子テーブルスキャン方式の値については、キャッシュ有効時においても同じ値をとる。

表6は、キャッシュ無効時における実行計画から読み出しページ数を抽出したものである。総読み出しページ数は問い合わせ処理全体で読み出されたページ数の総計である。実行計画の最初に表示される **Buffers** の値を総読み出しページ数とした。子テーブル読み出しページ数は **tbl_fheader_raXXX** の子テーブルのスキャン処理時に読み出されたページ数である。スキャンされる子テーブルが1個の場合はその子テーブルのスキャン処理中に表示される **Buffers** の値を、スキャンされる子テーブルが複数ある場合は **Parallel Append** 処理中に表示される **Buffers** の値を、それぞれ子テーブル読み出しページ数とした。hitは共有バッファから読み出されたページ数、readはディスクから読み出されたページ数をそれぞれ表す[30]。1ページは8KiBである[21]。

4.3 考察

キャッシュ無効時

■**ピンポイント検索とラフ検索** ピンポイント検索とラフ検索ではテーブル分割数が増えるにつれ実行時間が減少した。非分割時の実行時間と比べ前者は最大55%程、後者は最大83%程短縮された。実行時間が減少した原因の一つとして子テーブルスキャン時間の減少が考えられる。例えばパラメータ設定値が8の場合の **view_tomoe_ra006** と **view_tomoe_ra012** に対するピンポイント検索では、表4より実行時間が8,899.4msから8,516.0msに短縮されている。表5より両者のワーカー起動数、被スキャン子テーブル数、子テーブルスキャン方式は同一であり、実行時間の差が並列度の上昇やスキャン方式の違いにより生じたものではないことがわかる。一方、子テーブルスキャン時間は2,912.6msから2,471.0msに短縮されており、テーブル分割数が増えたことで子テーブルスキャン時間が減少したものと考えられる。

テーブル分割数が増えると前論文4.1節で述べたとおり、子テーブルのテーブルファイルに同じ分割範

囲のデータが集約され、1ページに含まれる検索条件に一致するデータの数が増加することで、読み出しページ数が減少すると考えられる。表6の子テーブル読み出しページ数を確認すると、分割キー (**crval1**) と検索条件 (**crval1,crval2**) が一致するピンポイント検索とラフ検索では分割数が増えるにつれページ数が減少している。一方、分割キー (**crval1**) と検索条件 (**date_obs**) が一致しないカレンダー検索では読み出しページ数が増加傾向にあり、テーブル分割によるデータの集約が確認できた。なお、キャッシュを削除しているにもかかわらず表6の **hit** (共有バッファから読み出されたページ数) が値を持っていることについては、PostgreSQLが起動した時点で自動的にページをキャッシュするためと考えられる。キャッシュを削除した状態でPostgreSQLを起動し、その前後の共有メモリの使用量をShellコマンドの **free** で観察すると、起動後の共有メモリの増加を確認できる。

ラフ検索の実行時間が180分割以上で増加した理由は、1ワーカーが処理する子テーブルの数が増加したためと考えられる。表5より、パラメータ設定値が8の場合の90, 180, 360分割時における被スキャン子テーブル数がそれぞれ4, 8, 15個であるのに対し、ワーカー起動数はそれぞれ3, 4, 1個である。被スキャン子テーブル数が倍増しているのに対し、ワーカー起動数は180分割時に1ワーカー増えたのみで、理由は不明だが360分割時には減少してしまっている。

■**カレンダー検索** カレンダー検索の実行時間が増加した理由は、被スキャン子テーブル数が増加したためと考えられる。カレンダー検索は **date_obs** 列を検索するSQLクエリである。したがって、**crval1** 列を分割キーとしたテーブルから構成される **view_tomoe_raXXX** に対する検索では、全ての子テーブルをスキャンしなければならない。また表5より、被スキャンテーブル数が倍増していくのに対し、ワーカー起動数が1個ずつしか増加しなかったことも、実行時間が増加した原因と考えられる。

■**最適なテーブル分割数** テーブルを分割すると検索条件に一致する子テーブルのみが処理対象となることで検索速度が高速化される。一方、細分化しすぎると多数の子テーブルが検索条件に一致するため、ワーカー起動数が不足し、検索速度が低下すると考えられる。したがって、ピンポイント検索やラフ検索のような領域検索では、テーブルの分割範囲が検索範囲程度となる分割数が適切であると考えられる。

例えばラフ検索では表4より90分割時の実行時間が最も短くなったが、ラフ検索の検索半径が検索座標

表4 view_tomoe_raXXXに対して実験用SQLクエリを実行した際のキャッシュ無効時並びに有効時における実行時間の平均値。実行時間の単位はms、括弧内の値は標準誤差。8, 16, 32, 64はパラレルクエリのパラメータ設定値。

ビュー	キャッシュ無効				キャッシュ有効			
	8	16	32	64	8	16	32	64
view_tomoe	13,830.7 (99.9)	13,738.1 (30.2)	13,813.1 (48.0)	13,673.4 (107.2)	90.6 (10.6)	80.4 (1.3)	88.2 (8.5)	90.2 (11.1)
view_tomoe_ra006	8,899.4 (71.3)	8,879.7 (85.0)	8,932.7 (51.4)	8,918.8 (96.4)	92.3 (11.0)	90.4 (11.3)	91.0 (11.0)	89.3 (11.6)
view_tomoe_ra012	8,516.0 (48.3)	8,388.3 (71.0)	8,400.6 (64.1)	8,593.5 (17.1)	90.7 (11.1)	89.8 (10.0)	90.3 (11.6)	112.0 (1.6)
view_tomoe_ra024	7,791.7 (51.9)	7,739.2 (60.2)	7,882.3 (38.6)	7,859.9 (27.6)	89.9 (9.7)	90.4 (10.8)	90.7 (10.6)	88.5 (11.0)
view_tomoe_ra036	6,894.5 (20.1)	6,916.2 (57.7)	6,905.2 (29.2)	6,915.3 (32.7)	76.6 (10.7)	92.1 (1.3)	71.3 (10.0)	74.8 (11.0)
view_tomoe_ra048	30,451.5 (125.8)	6,438.6 (225.7)	6,552.8 (10.3)	6,593.1 (14.5)	99.1 (14.1)	55.6 (1.8)	82.2 (9.0)	70.1 (1.3)
view_tomoe_ra090	6,182.9 (67.3)	6,339.7 (25.4)	6,354.5 (26.9)	6,303.0 (33.2)	74.6 (9.9)	77.1 (7.0)	67.1 (11.7)	56.5 (2.1)
view_tomoe_ra180	6,346.7 (41.2)	6,399.5 (35.1)	6,325.4 (125.6)	6,366.8 (15.8)	68.4 (8.2)	62.7 (12.6)	91.5 (2.1)	65.7 (4.6)
view_tomoe_ra360	5,550.2 (73.7)	5,555.4 (48.4)	5,608.0 (58.4)	5,586.1 (47.9)	55.3 (4.5)	50.4 (2.8)	59.1 (1.5)	46.9 (4.7)
view_tomoe	95,376.6 (115.1)	95,771.2 (99.0)	95,663.2 (261.6)	96,075.0 (182.4)	5,466.3 (98.2)	5,358.7 (86.5)	5,320.9 (109.2)	5,369.2 (98.0)
view_tomoe_ra006	28,550.1 (561.1)	28,819.3 (75.6)	28,270.6 (392.8)	29,009.0 (573.9)	7,248.2 (699.8)	7,108.4 (614.5)	7,567.8 (1,069.3)	7,302.7 (818.8)
view_tomoe_ra012	26,942.2 (1,272.1)	26,347.7 (298.0)	26,881.6 (741.7)	26,550.5 (196.1)	7,504.3 (241.2)	7,344.9 (195.1)	7,744.5 (392.9)	8,205.1 (135.4)
view_tomoe_ra024	23,819.0 (393.7)	24,526.7 (301.0)	23,878.2 (81.7)	23,915.7 (66.2)	4,384.9 (103.9)	4,355.2 (376.0)	5,183.4 (186.8)	4,170.6 (91.4)
view_tomoe_ra036	20,577.1 (319.7)	20,636.3 (273.0)	20,593.5 (128.4)	20,652.9 (197.4)	4,142.7 (368.8)	4,208.4 (236.1)	4,174.3 (300.0)	3,870.9 (383.6)
view_tomoe_ra048	79,404.9 (143.7)	25,575.8 (395.7)	26,094.7 (91.8)	25,432.6 (124.5)	5,893.7 (87.5)	11,706.4 (196.0)	11,405.3 (444.2)	10,903.5 (164.4)
view_tomoe_ra090	16,461.2 (73.5)	16,589.5 (40.3)	16,297.9 (157.1)	16,307.2 (201.1)	5,459.8 (425.1)	5,308.4 (205.8)	5,039.4 (36.6)	5,117.3 (218.1)
view_tomoe_ra180	29,731.2 (6,024.0)	21,295.9 (142.1)	21,218.9 (89.5)	21,481.5 (239.0)	2,203.6 (109.4)	7,596.4 (91.7)	7,755.2 (119.2)	7,744.9 (130.1)
view_tomoe_ra360	99,087.6 (227.5)	98,439.4 (194.9)	94,500.6 (238.5)	53,842.2 (2,217.9)	5,860.0 (50.6)	5,798.7 (81.0)	5,778.8 (38.4)	27,767.3 (307.1)
view_tomoe	1,134.8 (12.0)	1,170.1 (21.3)	1,229.7 (34.7)	1,158.3 (20.7)	322.0 (13.0)	306.4 (8.5)	299.9 (4.7)	315.2 (12.7)
view_tomoe_ra006	887.5 (9.6)	887.2 (13.7)	927.4 (27.7)	1,071.4 (215.7)	180.3 (8.7)	188.3 (9.1)	182.3 (7.2)	185.8 (6.1)
view_tomoe_ra012	859.2 (24.8)	849.8 (39.7)	872.9 (20.4)	854.0 (8.8)	165.1 (11.9)	164.8 (9.9)	173.2 (1.2)	165.6 (9.8)
view_tomoe_ra024	967.3 (37.2)	979.4 (6.6)	937.4 (32.9)	983.4 (25.5)	147.6 (6.1)	156.4 (6.7)	150.1 (8.3)	151.3 (4.9)
view_tomoe_ra036	1,189.5 (14.2)	1,124.3 (24.4)	1,161.7 (14.8)	1,154.3 (33.9)	143.0 (3.6)	140.7 (3.8)	149.3 (1.8)	145.5 (1.6)
view_tomoe_ra048	1,526.5 (27.1)	1,182.9 (20.0)	1,129.9 (15.4)	1,141.2 (18.7)	159.0 (5.6)	113.4 (6.5)	122.4 (5.2)	126.5 (1.0)
view_tomoe_ra090	2,355.6 (28.4)	1,908.3 (15.1)	1,953.9 (11.5)	1,869.5 (19.3)	159.8 (6.7)	127.6 (4.7)	119.4 (5.1)	115.4 (5.3)
view_tomoe_ra180	3,049.6 (16.2)	2,687.0 (43.7)	2,753.0 (32.3)	2,705.2 (41.1)	169.2 (2.5)	131.8 (4.0)	131.9 (2.3)	128.8 (5.4)
view_tomoe_ra360	5,440.5 (43.5)	4,576.3 (52.8)	4,567.2 (86.4)	4,388.7 (54.8)	217.6 (4.6)	179.7 (5.4)	173.9 (0.9)	168.6 (4.9)

表5 view_tomoe_raXXXに対して実験用SQLクエリを実行した際のキャッシュ無効時における実行計画から抽出した情報。8, 16, 32, 64はパラレルクエリのパラメータ設定値。ワーカー起動数, 被スキャン子テーブル数, 子テーブルスキャン方式の値についてはキャッシュ有効時と同じ値である。

ビュー	ワーカー起動数				被スキャン子テーブル数				子テーブルスキャン方式				子テーブルスキャン時間 (ms)			
	8	16	32	64	8	16	32	64	8	16	32	64	8	16	32	64
view_tomoe	1	1	1	1	1	1	1	1	PB	PB	PB	PB	5,528.8	5,764.8	5,841.2	5,460.3
view_tomoe_ra006	1	1	1	1	1	1	1	1	PB	PB	PB	PB	2,912.6	2,987.2	3,000.2	3,001.7
view_tomoe_ra012	1	1	1	1	1	1	1	1	PB	PB	PB	PB	2,471.0	2,438.1	2,389.4	2,480.9
view_tomoe_ra024	1	1	1	1	1	1	1	1	PB	PB	PB	PB	1,469.5	1,571.0	1,472.2	1,519.8
view_tomoe_ra036	2	2	2	2	2	2	2	2	PB	PB	PB	PB	1,066.3	1,019.0	681.8	1,143.1
view_tomoe_ra048	1	2	2	2	1	1	1	1	I	PB	PB	PB	18,824.3	326.6	677.2	662.7
view_tomoe_ra090	2	2	2	2	1	1	1	1	PB	PB	PB	PB	404.5	383.9	394.5	382.6
view_tomoe_ra180	2	2	2	2	2	2	2	2	PB	PB	PB	PB	372.6	375.9	339.2	319.1
view_tomoe_ra360	2	2	2	2	3	3	3	3	PB	PB	PB	PB	194.6	204.2	224.4	166.7
view_tomoe	1	1	1	1	1	1	1	1	I	I	I	I	49,250.5	49,395.4	49,499.4	49,671.9
view_tomoe_ra006	2	2	2	2	1	1	1	1	PB	PB	PB	PB	19,937.7	19,030.6	19,436.2	20,158.6
view_tomoe_ra012	2	2	2	2	2	2	2	2	PB	PB	PB	PB	17,603.6	16,259.2	16,081.9	16,306.2
view_tomoe_ra024	3	3	3	3	2	2	2	2	PB	PB	PB	PB	13,491.7	12,284.3	14,236.2	13,940.9
view_tomoe_ra036	3	3	3	3	3	3	3	3	PB	PB	PB	PB	10,036.6	10,228.2	10,272.5	9,970.3
view_tomoe_ra048	1	2	2	2	3	3	3	3	I	PB	PB	PB	43,110.4	14,441.3	14,400.6	13,805.3
view_tomoe_ra090	3	3	3	3	4	4	4	4	PB	PB	PB	PB	8,047.2	8,223.8	8,082.8	7,952.5
view_tomoe_ra180	4	4	4	4	8	8	8	8	PB	PB	PB	PB	11,209.9	8,901.5	8,895.1	9,035.4
view_tomoe_ra360	1	1	1	4	15	15	15	15	I	I	I+PB	I+PB	60,413.5	59,376.7	55,672.6	20,115.8
view_tomoe	2	2	2	2	1	1	1	1	PI	PI	PI	PI	698.9	667.7	730.6	727.2
view_tomoe_ra006	3	3	3	3	6	6	6	6	PI	PI	PI	PI	305.5	308.0	292.3	274.3
view_tomoe_ra012	4	4	4	4	12	12	12	12	PI	PI	PI	PI	301.9	292.1	275.9	252.0
view_tomoe_ra024	5	5	5	5	24	24	24	24	PI	PI	PI	PI	445.0	480.3	512.4	541.3
view_tomoe_ra036	6	6	6	6	36	36	36	36	PI	PI	PI	PI	636.5	666.5	646.6	651.1
view_tomoe_ra048	4	6	6	6	48	48	48	48	PB+PI	PB+PI	PB+PI	PB+PI	889.9	785.4	672.9	748.1
view_tomoe_ra090	4	7	7	7	90	90	90	90	PI	PI	PI	PI	1,396.4	1,281.0	1,498.7	1,314.9
view_tomoe_ra180	4	8	8	8	180	180	180	180	PI	PI	PI	PI	2,367.1	2,143.0	2,158.6	2,170.5
view_tomoe_ra360	4	9	9	9	360	360	360	360	PB+PI	PB+PI	PB+PI	PB+PI	4,929.1	4,105.2	4,206.5	3,877.0

小澤武揚, 他

表6 view_tomoe_raXXXに対してEXPLAIN(ANALYZE,BUFFERS)句を冠した実験用SQLクエリを実行した際のキャッシュ無効時における実行計画における読み出しページ数。8, 16, 32, 64はパラレルクエリのパラメータ設定値。hitは共有バッファから読み出されたページ数, readはディスクから読み出されたページ数。

ビュー	総読み出しページ数						子テーブル読み出しページ数					
	8		16		32		64		8		16	
	hit	read	hit	read	hit	read	hit	read	hit	read	hit	read
ビュー												
view_tomoe	16,059	6,497	16,059	6,497	16,059	6,497	16,059	6,497	0	4,168	0	4,168
view_tomoe_ra006	16,055	6,472	16,055	6,472	16,055	6,472	16,055	6,472	0	4,139	0	4,139
view_tomoe_ra012	16,055	6,477	16,055	6,477	16,055	6,477	16,055	6,477	0	4,144	0	4,144
view_tomoe_ra024	16,055	6,431	16,055	6,431	16,055	6,431	16,055	6,431	0	4,098	0	4,098
view_tomoe_ra036	16,069	6,461	16,069	6,461	16,069	6,461	16,069	6,461	6	4,128	6	4,128
view_tomoe_ra048	16,063	6,380	16,063	6,380	16,063	6,380	16,063	6,380	0	4,047	0	4,047
view_tomoe_ra090	16,063	6,188	16,063	6,188	16,063	6,188	16,063	6,188	0	3,855	0	3,855
view_tomoe_ra180	16,070	6,060	16,070	6,060	16,070	6,060	16,070	6,060	7	3,727	7	3,727
view_tomoe_ra360	16,075	5,980	16,075	5,980	16,075	5,980	16,075	5,980	12	3,647	12	3,647
view_tomoe	413,552	81,801	413,552	81,801	413,552	81,801	413,552	81,801	17,337	73,603	17,337	73,603
view_tomoe_ra006	397,762	72,331	397,762	72,331	397,762	72,331	397,762	72,331	1,553	64,130	1,553	64,130
view_tomoe_ra012	397,773	65,761	397,773	65,761	397,773	65,761	397,773	65,761	1,564	57,560	1,564	57,560
view_tomoe_ra024	398,385	59,980	398,385	59,980	398,385	59,980	398,385	59,980	2,175	51,779	2,175	51,779
view_tomoe_ra036	398,391	59,110	398,391	59,110	398,391	59,110	398,391	59,110	2,181	50,909	2,181	50,909
view_tomoe_ra048	397,781	59,195	397,781	59,195	397,781	59,195	397,781	59,195	1,572	50,994	1,572	50,994
view_tomoe_ra090	398,397	59,260	398,397	59,260	398,397	59,260	398,397	59,260	2,187	51,059	2,187	51,059
view_tomoe_ra180	399,041	61,030	399,041	61,030	399,041	61,030	399,041	61,030	2,830	52,829	2,830	52,829
view_tomoe_ra360	399,095	64,735	399,095	64,735	399,095	64,735	399,095	64,735	2,884	56,534	2,884	56,534
view_tomoe	338,331	13,047	338,333	13,047	338,334	13,047	338,335	13,047	256	11,584	258	11,584
view_tomoe_ra006	338,153	13,069	338,155	13,068	338,148	13,068	338,148	13,068	82	11,604	84	11,603
view_tomoe_ra012	338,099	13,086	338,097	13,085	338,096	13,085	338,100	13,085	27	11,621	25	11,620
view_tomoe_ra024	338,106	13,132	338,102	13,132	338,104	13,131	338,100	13,134	33	11,667	29	11,666
view_tomoe_ra036	338,106	13,188	338,101	13,188	338,100	13,187	338,104	13,186	32	11,723	27	11,722
view_tomoe_ra048	338,115	13,229	338,117	13,229	338,117	13,230	338,113	13,231	41	11,764	43	11,765
view_tomoe_ra090	338,149	13,389	338,146	13,391	338,147	13,389	338,146	13,391	74	11,924	71	11,926
view_tomoe_ra180	338,221	13,737	339,699	12,271	338,234	13,738	338,228	13,737	146	12,272	158	12,273
view_tomoe_ra360	338,388	14,366	338,398	14,368	338,387	14,368	338,399	14,369	313	12,901	321	12,903

点を中心とした半径6度であるのに対し、90分割時のテーブル分割範囲は表3より赤経4度毎であった。表5より、12, 24, 48, 90, 180, 360分割時のラフ検索の被スキャン子テーブル数はそれぞれ2, 2, 3, 4, 8, 15個であり、テーブルの分割範囲が検索半径を下回ると処理対象の子テーブル数が急増することがわかる。ただし、本実験ではテーブルを赤経方向に分割しているため、検索座標点が極に近づくほど被スキャン子テーブル数は増加する。ラフ検索の実験用SQLクエリの検索座標点が赤緯-5度の位置にあったことから、前述した関係が成り立った。極に近づくほど被スキャン子テーブル数が増加する現象は赤道方向分割の弱点であり、これを回避するには赤緯や他の方法による分割を考える必要があるだろう。

■**最適なパラレルクエリのパラメータ設定値** いずれの実験用SQLクエリにおいても、パラレルクエリのパラメータ設定値の違いによる実行時間の系統的な変化はみられなかった。ワーカー起動数はプランナによって自動的に決定されるが、表5に示されるとおり、そのほとんどはデフォルトのパラメータ設定値である8以下の値であった。したがってパラメータ設定値を16, 32, 64と増やしてもワーカー起動数が変化しなかったものと考えられる。一方、パラメータ設定値の変更がワーカー起動数に影響を与えていると思われる例外も見受けられた。例えば、パラメータ設定値が64の場合のview_tomoe_ra360に対するラフ検索は、ワーカー起動数が他のパラメータ設定値の4倍多く、実行時間は半減している。ワーカー起動数が変化した原因は不明であるが、パラメータ設定値により抑制された可能性がある。そのため計算機資源に鑑み、パラメータ設定値は大きめの値にすべきと考える。

■**SMOKA/Tomo-e Gozenシステムへの応用** SMOKA/Tomo-e Gozenシステムでは、実行時間の長いキャッシュ無効時のピンポイント検索とラフ検索の検索速度を高速化するために、crval1を分割キーとしてtmq_kisoテーブルを90分割し、パラレルクエリのパラメータ設定値をCPUコア数と同数の24程度にすべきと考える。

■**3TB HDDと8TB HDDの比較** 表4のパラメータ設定値が8の場合のview_tomoeに対する実行時間と、前論文4.2節の実験におけるパラメータ設定値が8の場合のview_tomoeに対する実行時間（前論文の表16, 表17, 表18を参照）を比較すると、ピンポイント検索は26%程、ラフ検索は17%程、カレンダー検索は57%程、本実験の実行時間が長かった。両実験のデータベース

の設定やテーブル定義、SQLクエリの内容、ワーカー起動数等は同じであり、異なる要素はHDDの種類のみである（第3節参照）。3TB HDDと8TB HDDは容量以外の諸元は同じであるが、3TBのHDDの方が8TBのHDDよりも読み出し性能が3割程悪いのかもしれない。

キャッシュ有効時

■**ピンポイント検索とカレンダー検索** ピンポイント検索とカレンダー検索ではテーブル分割数と実行時間との間に相関がみられたが、実行時間の変化量は数十msと小さい。キャッシュ無効時は数百から数千msの変化量があったことから、これらの検索についてはキャッシュ無効時の実行時間の長短によってテーブル分割数を選択すべきと考える。

■**ラフ検索** ラフ検索ではテーブル分割数によって実行時間が非分割テーブルから構成されるview_tomoeより短くなる場合と長くなる場合があった。本実験で得られた情報からはその理由はわからなかった。表4より、キャッシュ有効時の実行時間は多くの場合4,000msから8,000msの値をとる一方、キャッシュ無効時は16000msから99000msまでの値をとる。したがってラフ検索においても、キャッシュ無効時の実行時間の長短によってテーブル分割数を選択すべきと考える。

■**最適なパラレルクエリのパラメータ設定値** いずれの実験用SQLクエリにおいても、パラレルクエリのパラメータ設定値の違いによる実行時間の系統的な変化はみられなかった。ただしキャッシュ無効時と同様、パラメータ設定値がワーカー起動数と子テーブルスキャン方式に影響を与えることで、実行時間が変化したと思われる例外が見受けられた。例えばview_tomoe_ra360に対するカレンダー検索では、表5よりパラメータ設定値が8の場合のワーカー起動数は4であるが、16以上の場合は9である。実行時間を表4より確認すると、パラメータ設定値が8の場合は16以上の場合よりも40-50ms長い。

5 SSD実験

テーブルファイルとインデックスファイルをSSD上に配置した場合の実験用SQLクエリの実行時間が、HDD上に配置した場合の実行時間と比べ、どの程度高速化されるのか調査した。調査には前論文[17] 3節の実験手法を用いた。当該実験は入力フレーム数の異

表7 SSD実験用のテーブルとビューの一覧.

メタ情報テーブル	管理情報テーブル	実験用ビュー	フレーム数
tbl_fheader_00050k_ssd	tbl_filemng_00050k_ssd	view_tomoe_00050k_ssd	50,000
tbl_fheader_00100k_ssd	tbl_filemng_00100k_ssd	view_tomoe_00100k_ssd	100,000
tbl_fheader_00200k_ssd	tbl_filemng_00200k_ssd	view_tomoe_00200k_ssd	200,000
tbl_fheader_00400k_ssd	tbl_filemng_00400k_ssd	view_tomoe_00400k_ssd	400,000
tbl_fheader_00800k_ssd	tbl_filemng_00800k_ssd	view_tomoe_00800k_ssd	800,000
tbl_fheader_01600k_ssd	tbl_filemng_01600k_ssd	view_tomoe_01600k_ssd	1,600,000
tbl_fheader_03200k_ssd	tbl_filemng_03200k_ssd	view_tomoe_03200k_ssd	3,200,000
tbl_fheader_06400k_ssd	tbl_filemng_06400k_ssd	view_tomoe_06400k_ssd	6,400,000
tbl_fheader_12800k_ssd	tbl_filemng_12800k_ssd	view_tomoe_12800k_ssd	12,800,000
tbl_fheader_28000k_ssd	tbl_filemng_28000k_ssd	view_tomoe_28000k_ssd	27,918,958

なる複数のテーブルを用意し、それぞれのテーブル並びにビューに対する実験用SQLクエリの実行時間を測定するものである。前論文ではHDD上にテーブルファイルとインデックスファイルを配置して実験をおこなっていたが、これをSSDに変更して実行時間を測定し、その結果を比較する。

5.1 実験方法

表7は作成したSSD実験用のテーブルとビューの一覧である。tbl_fheader_XXXXXk_ssdとtbl_filemng_XXXXXk_ssdには、フレーム数列に示した数のスタック済みデータのメタ情報と管理情報がそれぞれ入力されている。tbl_fheader_28000k_ssdとtbl_filemng_28000k_ssdに約2800万フレームのスタック済みデータの情報をファイル名昇順で入力した後、両テーブルからtbl_fheader_XXXXXk_ssdとtbl_filemng_XXXXXk_ssdへ、frame_id昇順でデータをINSERTした。tbl_fheader_XXXXXk_ssdとtbl_filemng_XXXXXk_ssdから構成されるview_tomoe_XXXXXk_ssdをそれぞれ作成した。

実験用計算機に搭載している7本のテーブル空間用HDDの内、hdd7をSSDに交換した。SSD上にテーブル空間を作成し、テーブル並びにインデックス作成時に同テーブル空間を指定することで、テーブルファイルとインデックスファイルをSSD上に配置した。

5.2 実験結果

図4は、キャッシュ無効時並びに有効時における実行時間の平均値 (Time) に対するフレーム数 (Number of Frames) のグラフである。実行時間の単位はms、誤差棒は実行時間の標準誤差を表す。比較のため、HDDを使用したtbl_fheader_XXXXXk並びにview_tomoe_

XXXXXkに対する実行時間を、前論文の表6, 7, 8から引用しプロットしている。キャッシュ無効時はテーブル並びにビューに対する実験用SQLクエリの実行時間が、HDD上に配置した場合よりも短縮されることがわかった。キャッシュ有効時はいずれの実験用SQLクエリにおいても実行時間が短縮されなかった。表8は図4の作成に用いた数値データである。

5.3 考察

■**キャッシュ無効時** 表8のrより、SSDを使用することでテーブルに対する実行時間が、ピンポイント検索では最大97%程、ラフ検索では最大91%程短縮された。カレンダー検索ではフレーム数が50kから800kの時は最大98%程、1600kから12800kの時は最大47%程短縮されたが、28000kの時は9%程長くなった。28000kの時のHDDを使用した場合のテーブルのスキャン方式がIndex Scanである一方、SSDを使用した場合のテーブルのスキャン方式はBitmap Heap Scanであったことから、テーブルのスキャン方式の違いで実行時間が長くなったものと考えられる。

ビューに対する実行時間は、SSDを使用したことで、ピンポイント検索では最大98%程、ラフ検索では最大92%程短縮された。カレンダー検索ではフレーム数が50kから800kの時は最大96%程、3200kから28000kの時は最大38%程短縮されたが、1600kの時は外れ値となった上に4%程しか短縮されなかった。ビューを構成するメタ情報テーブルと管理情報テーブルの結合方式を実行計画から調べると、SSDとHDDの双方でNested Loopが使われていた。しかしフレーム数が1600kの時はHash Joinが使われており、結合方式の違いが外れ値の原因と推測される。

■**キャッシュ有効時** 実験用SQLクエリの総読み出

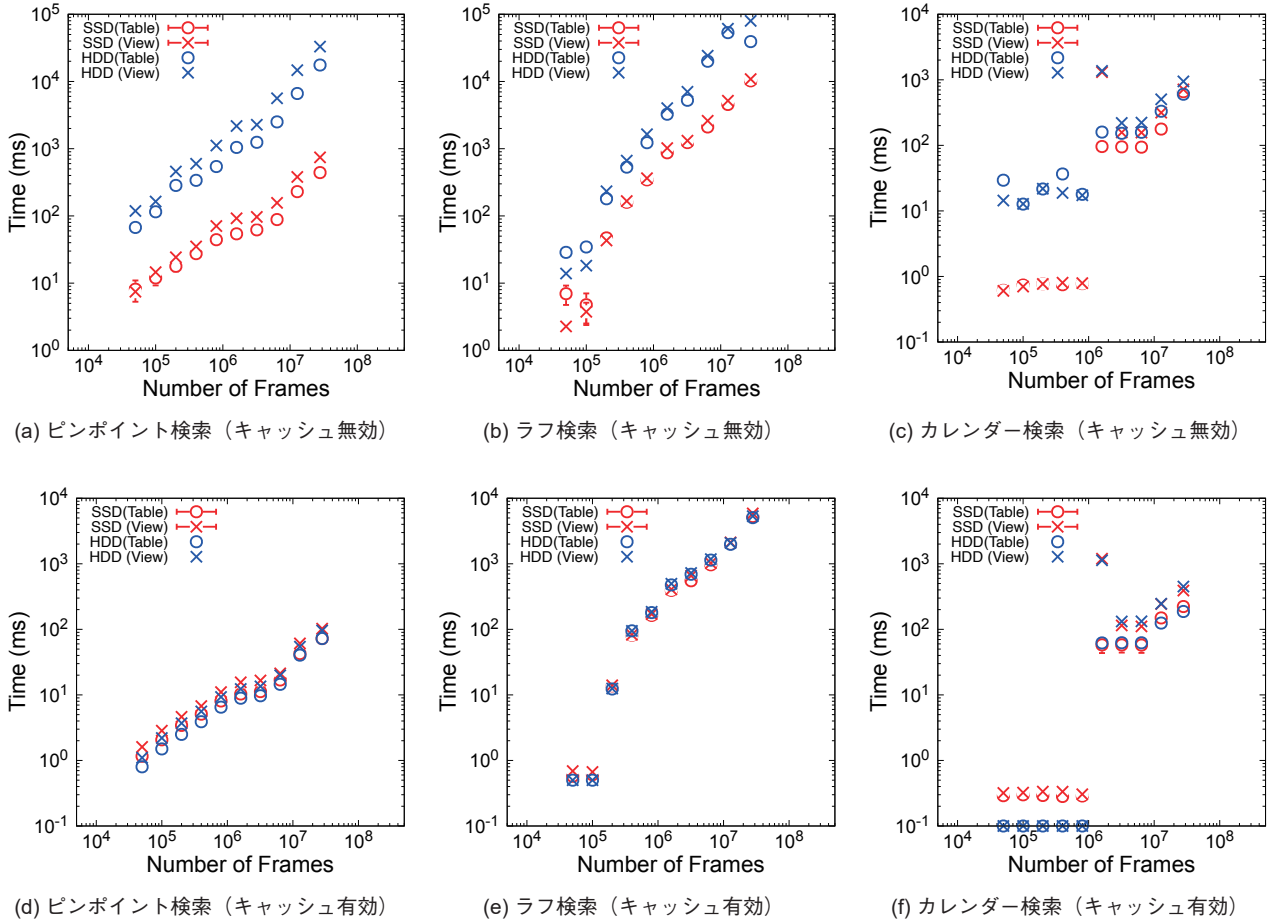


図4 tbl_fheader_XXXXXk_ssd並びにview_tomoe_XXXXXk_ssdに対して実験用SQLクエリを実行した際の、キャッシュ無効時並びに有効時における実行時間の平均値に対するフレーム数。実行時間の単位はms、誤差棒は実行時間の標準誤差。比較のためのHDDを使用したtbl_fheader_XXXXXk並びにview_tomoe_XXXXXkに対する実行時間を、前論文の表6, 7, 8から引用しプロットしている。凡例のSSD(Table)並びにSSD(View)はtbl_fheader_XXXXXk_ssd並びにview_tomoe_XXXXXk_ssdに対する実行時間、HDD(Table)並びにHDD(View)はtbl_fheader_XXXXXk並びにview_tomoe_XXXXXkに対する実行時間を表す。

しページ数は表6より最大でも50万ページ (~3.8 GiB) 程度であるため、キャッシュ有効時にディスクアクセスは発生しない。したがってキャッシュ無効時と有効時の実行時間の差は、SSDとHDDのI/O性能の違いによって生じたものと考えられる。

■SMOKA/Tomo-e Gozenシステムへの応用 SMOKA/Tomo-e Gozenシステムにおいては、キャッシュ無効時の実行時間を最大2桁高速化できることからテーブルファイルとインデックスファイルを配置するためのSSDを積極的に導入すべきと考える。

6 PostGIS実験

PostGISのGISオブジェクトのGEOGRAPHY型データのうち、球面上の点を表す空間型はPOINTデータと

呼ばれる[31]。本実験では、従来のテーブルのra, decの組及びcrval1, crval2の組をGEOGRAPHY型POINTデータに変換したものにR-tree空間インデックス[19]を追加して、それらGEOGRAPHY型POINTデータを使ったピンポイント検索及びラフ検索の実行時間を測定した。検索範囲の絞り込みにPostGIS関数を利用する場合と利用しない場合の実行時間の比較もおこなった。本実験ではPostGISのみの効果を確認するためテーブル分割、パラレルクエリ、SSDは利用していない。

6.1 実験方法

球体モデルの登録

PostGISの拡張機能をデータベースに追加することで、地球の形状を近似する多くの楕円体モデルが登録されたspatial_ref_sysテーブルが当該データベースに

表8 tbl_fheader_XXXXXk_ssd並びにview_tomoe_XXXXXk_ssdに対して実験用SQLクエリを実行した際の、キャッシュ無効時並びに有効時における実行時間の平均値. 実行時間の単位はms, 括弧内の値は標準誤差. rはHDDを使用した場合の実行時間に対するSSDを使用した場合の実行時間の比. スキャン方式はtbl_fheader_XXXXXk_ssdに対して実行されたスキャンの方式 (I: Index Scan, B: Bitmap Heap Scan).

フレーム数	tbl_fheader_XXXXXk_ssd			view_tomoe_XXXXXk_ssd			r	スキャン方式
	キャッシュ無効	r	キャッシュ有効	キャッシュ無効	r	キャッシュ有効		
50k	8.1 (2.8)	0.12	1.2 (0.1)	7.5 (0.1)	0.06	1.6 (0.1)	1.44	B
100k	12.0 (2.7)	0.10	2.1 (0.0)	14.6 (1.6)	0.09	2.9 (0.2)	1.31	B
200k	17.8 (3.0)	0.06	3.4 (0.2)	24.3 (1.3)	0.05	4.6 (0.1)	1.27	B
400k	27.5 (2.9)	0.08	5.1 (0.3)	35.1 (2.3)	0.06	6.8 (0.3)	1.20	B
800k	44.3 (3.3)	0.08	8.1 (0.4)	70.7 (1.4)	0.06	11.0 (0.7)	1.19	I
1600k	54.1 (3.1)	0.05	10.3 (0.8)	92.1 (1.6)	0.04	15.6 (0.8)	1.27	I
3200k	62.1 (1.8)	0.05	11.2 (0.6)	96.6 (1.3)	0.04	16.6 (1.1)	1.23	I
6400k	88.3 (2.1)	0.04	16.9 (1.0)	156.4 (3.4)	0.03	21.3 (2.6)	1.07	I
12800k	229.8 (3.0)	0.03	42.7 (4.0)	380.0 (3.8)	0.03	61.0 (6.8)	1.12	I
28000k	441.3 (4.7)	0.03	72.4 (8.6)	743.5 (15.5)	0.02	102.5 (10.8)	1.07	I
50k	7.0 (2.2)	0.24	0.6 (0.0)	2.3 (0.0)	0.16	0.7 (0.0)	1.36	B
100k	4.8 (2.3)	0.14	0.6 (0.0)	3.8 (1.4)	0.21	0.7 (0.0)	1.35	I
200k	47.1 (0.5)	0.26	13.7 (0.4)	43.1 (1.9)	0.19	14.0 (0.6)	1.12	I
400k	159.5 (9.0)	0.30	81.3 (9.3)	166.4 (7.3)	0.25	82.6 (9.1)	0.87	B
800k	344.0 (19.0)	0.28	163.0 (10.7)	363.9 (23.1)	0.22	170.7 (10.8)	0.92	B
1600k	869.4 (44.5)	0.27	393.0 (49.5)	1,021.1 (45.2)	0.25	407.8 (51.8)	0.82	B
3200k	1,232.6 (85.1)	0.23	551.0 (79.1)	1,316.3 (68.4)	0.19	656.3 (81.9)	0.91	B
6400k	2,104.5 (85.3)	0.11	965.5 (87.6)	2,601.4 (115.4)	0.11	1,024.3 (87.5)	0.87	B
12800k	4,550.1 (220.9)	0.09	2,033.4 (7.3)	5,171.6 (166.2)	0.08	2,123.1 (8.9)	1.02	B
28000k	10,229.2 (369.7)	0.26	5,265.6 (25.3)	10,852.4 (270.6)	0.14	5,867.7 (243.2)	1.10	B
50k	0.6 (0.0)	0.02	0.3 (0.0)	0.602 (0.0)	0.04	0.3 (0.0)	3.35	I
100k	0.7 (0.0)	0.06	0.3 (0.0)	0.7 (0.0)	0.05	0.3 (0.0)	3.16	I
200k	0.8 (0.0)	0.04	0.3 (0.0)	0.8 (0.0)	0.04	0.3 (0.0)	2.62	I
400k	0.8 (0.0)	0.02	0.3 (0.0)	0.8 (0.0)	0.04	0.3 (0.0)	2.61	I
800k	0.8 (0.0)	0.04	0.3 (0.0)	0.8 (0.0)	0.05	0.3 (0.0)	2.89	I
1600k	96.0 (1.0)	0.60	58.0 (14.4)	1,309.1 (3.3)	0.96	1,196.2 (12.0)	1.06	I
3200k	94.8 (1.7)	0.62	57.9 (13.7)	157.5 (1.0)	0.72	114.7 (10.3)	0.87	I
6400k	93.9 (1.1)	0.59	57.6 (13.9)	156.7 (5.1)	0.70	110.9 (11.2)	0.84	I
12800k	177.2 (3.4)	0.53	149.3 (2.8)	315.3 (2.3)	0.62	243.5 (11.4)	0.99	I
28000k	658.4 (27.8)	1.09	222.9 (19.9)	713.9 (6.9)	0.75	390.9 (49.4)	0.87	B

追加される。天球面上の座標を扱う場合には、扁平率が0の楕円体 (=球体) モデルが必要であるが、そのような楕円体モデルは spatial_ref_sys テーブルには登録されていない。そのため以下のSQL文を使い、spatial_ref_sys テーブルに球体モデルをSRID(Spatial Reference Identifier)=40000として登録した。

```
INSERT INTO spatial_ref_sys VALUES(
    40000,
    'ME',
    1,
    'GEOGCS[
        "Normal Sphere",
        DATUM[
            "unknown",
            SPHEROID["sphere",3437,0]
        ],
        PRIMEM[
            "Greenwich",
            0
        ],
        UNIT[
            "degree",
            0.0174532925199433
        ]
    ],
    '+proj=longlat +ellps=sphere +no_defs'
);
```

GEOGRAPHY 型 POINT データへの変換関数

赤経・赤緯座標 (ra,dec 及び crval1,crval2) の組を GEOGRAPHY 型 POINT データへ変換するためのユーザ定義関数 GISpoint を作成した。以下がその定義式である。

```
CREATE FUNCTION GISpoint(
    radeg NUMERIC,decdeg NUMERIC
) RETURNS GEOGRAPHY(POINT, 40000) AS $$
BEGIN
    RETURN ST_GeographyFromText(
        'SRID=40000; POINT(' ||
        CAST(radeg AS TEXT) ||
        ' ' ||
        CAST(decdeg AS TEXT) ||
        ')'
    );
END
```

```
$$ IMMUTABLE LANGUAGE PLPGSQL;
```

テーブルとビューの作成

第4で使用した tbl_fheader に GEOGRAPHY 型 POINT データの列を二つ追加した tbl_fheader_postgis を作成した。追加した列は、ra, dec の組及び crval1, crval2 の組をそれぞれユーザ定義関数 GISpoint を用いて GEOGRAPHY 型 POINT データに変換したものであり列名はそれぞれ radec_point, cval_point とした。これらの列には以下のコマンドを用いて GiST (Generalized Search Trees) インデックスを追加した。PostGIS は GiST 上で実装している R-tree インデックスを空間データのインデックスに使用している [32]。

```
CREATE INDEX ON tbl_fheader_postgis
USING GIST (radec_point);
CREATE INDEX ON tbl_fheader_postgis
USING GIST (crval_point);
```

これらのインデックスに加え、tbl_fheader と同様のインデックスを作成した。インデックス作成後に、クエリプランの最適化に使うテーブル統計情報の収集のため VACUUM ANALYZE tbl_fheader_postgis を実行した。

tbl_fheader_postgis から view_tomoe_postgis を作成した。ビューの定義は、前論文 [17] の補遺 A の SQL 3 の tmq_kiso を tbl_fheader_postgis に置き換え、b.publictime, の行の上に a.crval_point, 及び a.radec_point, が挿入されたものである。

クエリ実行

ピンポイント検索には、前論文の補遺 A の SQL 4 と同等の SQL クエリを使用した。ただし、FROM 句でテーブルに対する検索には tbl_fheader_postgis を、ビューに対する検索には view_tomoe_postgis を使用し、WHERE 句での検索範囲絞り込みの条件式は

```
crval_point && ST_MakeEnvelope(
    9.664629665561543,
    40.50208333333333,
    11.70470366777179,
    42.03541666666666,
    40000
)
```

とした。ST_MakeEnvelope [33] は、球面座標の最小値と最大値を指定することで矩形ポリゴンの領域を定義

する PostGIS 関数である。ただしこの条件式は、前論文の SQL 4 で使用している不等号による条件式

```
crval1 >= 9.664629665561543 AND
crval1 <= 11.70470366777179 AND
crval2 >= 40.50208333333333 AND
crval2 <= 42.03541666666666
```

よりも少し大きめの天球面の領域を含み、より多くの行（前者が4,365行に対し後者は3,676行）が検索にヒットする。比較のため、後者の条件式でも同じテーブルとビューに対して検索する実験をおこなった。

ラフ検索は、前論文の補遺 A の SQL 5 と同等の SQL クエリを使用した。ただし、FROM 句で Cone Search [34] をおこなう fconesearchtmqsmoka 関数を指定せず、テーブルに対する検索では tbl_fheader_postgis を、ビューに対する検索では view_tomoe_postgis を指定した。WHERE 句は

```
ST_DWITHIN(
  radec_spoint,
  ST_GeographyFromText(
    'SRID=40000;
    POINT(
      83.822041666666668
      -5.3910833333333334
    )'
  ),
  667169
) AND crval_spoint && ST_MakeEnvelope(
  9.664629665561543,
  40.50208333333333,
  11.70470366777179,
  42.03541666666666,
  40000
)
```

とした。ST_DWITHIN [35] は Cone Search を行う PostGIS 関数である。ここでも検索範囲の絞り込みに ST_MakeEnvelope を用いた。さらに、ピンポイント検索と同様に比較のため検索範囲の絞り込みに不等号による条件式を用いた実験もおこなった。

前論文ではテーブルに対して VACUUM ANALYZE を実行していなかった。VACUUM ANALYZE を実行した tbl_fheader_postgis と実験結果を比較するため、tbl_fheader に対して VACUUM ANALYZE を実行したうえで、テーブル分割せずパラレルクエリ非使用で PostGIS 非使用の場合のピンポイント検索とラフ検索の実行時間を改めて測定した（下記実験結果の非 PostGIS）。なお ANALYZE の有無による実行時間の違いについては本論文の補遺 A を参照のこと。

6.2 実験結果

実験結果を表9に示した。WHERE 句で検索範囲の絞り込みに PostGIS 関数 ST_MakeEnvelope による条件式を使用した場合の実験を PostGIS(ST) と表し、不等号による条件式を使用した場合の実験を PostGIS(noST) と表した。

各検索の実行計画で使用されるスキャン方式や結合方式等の項目は、非 PostGIS、PostGIS(ST)、PostGIS(noST) の間で同じであった。ピンポイント検索のテーブルに対する検索では、スキャン方式は WHERE 句で使用している列（crval1 と crval2、または crval_spoint）に対する Index Scan であった。ビューに対する検索では同スキャン方式に tbl_filemng テーブルの主キーに対する Index Scan が加わった。結合方式は Nested loop であった。ラフ検索のテーブルに対する検索では、スキャン方式は Bitmap Heap Scan と、WHERE 句並びに Cone Search 関数で使用している列に対する Bitmap Index Scan であった。ビューに対する検索では、同スキャン方式に tbl_filemng の主キーに対する Index Scan が加わった。結合方式は Nested loop であった。

表9 PostGIS 実験結果。実行時間の単位は ms であり、括弧内の値は標準誤差である。

		テーブル				ビュー			
		キャッシュ無効		キャッシュ有効		キャッシュ無効		キャッシュ有効	
ピンポイント検索	非 PostGIS	22,932.3	(12.9)	73.4	(0.2)	39,621.6	(94.1)	94.4	(0.3)
	PostGIS(ST)	23,254.7	(21.7)	38.9	(0.2)	63,951.7	(114.2)	72.4	(0.5)
	PostGIS(noST)	23,207.8	(4.5)	74.3	(0.4)	58,184.1	(78.8)	96.3	(0.5)
ラフ検索	非 PostGIS	20,675.0	(106.5)	5,228.1	(1.4)	42,076.2	(97.1)	5,586.1	(0.9)
	PostGIS(ST)	45,219.1	(64.1)	495.8	(1.6)	70,394.4	(95.7)	810.6	(1.6)
	PostGIS(noST)	27,050.8	(67.2)	809.8	(0.8)	52,918.8	(34.2)	1,041.2	(0.8)

6.3 考察

ピンポイント検索

■**キャッシュ無効時** テーブルに対する検索では、非PostGIS, PostGIS(ST), PostGIS(noST)の間で実行時間に大きな差はみられなかった。

ビューに対する検索では、非PostGISの場合が最も実行時間が短く、PostGIS(ST)の場合が最も実行時間が長く非PostGISの場合の1.6倍であった。WHERE句で使用している列に対するIndex Scanにかかる時間が非PostGISでの24,667msに比べて、PostGIS(ST)では35,736msと1.4倍かかっている。また、tbl_filemngの参照にかかる時間が非PostGISでの15,000msに比べて、PostGIS(ST)では28,000msと1.9倍かかっている。R-tree空間インデックスが使用されないPostGIS(noST)についても、非PostGISとの比較で同様の傾向がみられる。したがって、PostGIS利用時に非PostGISよりも実行時間が長くなる理由はR-tree空間インデックスの使用とは関係がなく、GEOGRAPY型データを含むテーブルを参照することに関係すると考えられるが、その原因は不明である。なお、キャッシュ無効時のBUFFERSの読み取りページ数は、Index Scan時には非PostGISで4,168ページ、PostGIS(ST)で4,582ページである。tbl_filemngの参照時には非PostGISで2,329ページ、PostGIS(ST)で2,672ページであり、読み取りページ数についても上記の実行時間増大の要因とは言えない。

■**キャッシュ有効時** テーブルに対する検索では、PostGIS(ST)の実行時間がその他と比べて47%程短縮された。実行時間のうち、Index Scanにかかる時間は非PostGISの66.0msと比べて、PostGIS(ST)では28.5msであり、キャッシュ有効時にはR-tree空間インデックスの効果が現れたと考えられる。なお、PostGIS(noST)ではIndex ScanにおいてR-tree空間インデックスは使用されず、B-treeインデックス (crval1とcrval2の複合インデックス) が使用され、非PostGISと同程度の時間がかかった。

ビューに対する検索では、PostGIS(ST)の実行時間が非PostGISの場合に比べ23%程短縮され、実行時間が最も短かった。実行時間のうち、Index Scanにかかる時間は非PostGISの47.6msと比べてPostGIS(ST)では12.5msであり、R-tree空間インデックスを使ったことによる効果と考えられる。

■**SMOKA/Tomo-e Gozenシステムへの応用** SMOKA/Tomo-e Gozenシステムの運用ではビューを検索対象としており、キャッシュ無効時にはPostGIS利用によ

り非PostGISの場合より実行時間が長くなる。そして、キャッシュ有効時にはPostGIS(ST)の利用で非PostGISの場合より実行時間が短くなる。しかし、現在のデータ総数では20ms程度しか時間短縮にならず、今後データ総数が増えて現在の10倍になった場合にも仮にキャッシュが有効であっても200ms程度の時間短縮にしかならないと考えられる(前論文の3.3章でピンポイント検索では実行時間はデータ総数に比例することが示された)。したがって、SMOKA/Tomo-e Gozenシステムのピンポイント検索にはPostGISの利用はあまり有効ではないと考えられる。

ラフ検索

■**キャッシュ無効時** テーブルとビューの双方に対する検索において、非PostGISの場合に最も実行時間が短く、PostGIS(ST)の場合に最も実行時間が長くなり、非PostGISの場合の2.2倍(テーブル)及び1.7倍(ビュー)となった。Bitmap Index Scanにかかる時間がこの時間差に反映されている。テーブルに対する検索では、検索範囲絞り込みの列 (crval1,crval2, またはcrval_spoint) に対するBitmap Index Scanにかかる時間は非PostGISで509ms、PostGIS(ST)では18,646msであり、Cone Search関数に使う列 (x,y,zまたはradec_spoint) に対するBitmap Index Scanにかかる時間は非PostGISで941ms、PostGIS(ST)では8,924msであった。ビューに対する検索についても数%以内で同程度の値であった。GEOGRAPHY型データに対するBitmap Index Scanは、通常の型のデータに対するものよりも非常に時間がかかると考えられる。キャッシュ無効時のBUFFERSの読み取りページ数は、テーブルに対する検索とビューに対する検索で同じ値であった。検索範囲絞り込みの列のBitmap Index Scan時については、非PostGISで6,738ページ、PostGIS(ST)では3,662ページである。Cone Search関数に使う列のBitmap Index Scan時については、非PostGISで16,351ページ、PostGIS(ST)では1,705ページである。PostGIS(ST)のほうが読み取りページ数が少なく、Bitmap Index Scanにかかる時間の差は読み取りページ数では説明できない。なお、PostGIS(noST)の場合には実行時間が非PostGISの場合の1.3倍であり、PostGIS(ST)ほどは長くない。これは、検索範囲絞り込みの列としてcrval1とcrval2を用いており、これら列に対するBitmap Index Scanにかかる時間が505msと、非PostGISの場合とほぼ同じ値であるためである。

■**キャッシュ有効時** PostGIS(ST)の場合に非PostGISと比べて顕著に実行時間が短くなり、テーブルに対す

る検索では非PostGISの場合に比べ90%程、ビューに対する検索では85%程短縮された。キャッシュが有効になることでテーブルに対する検索のBitmap Index Scanに要する時間が、crval_pointに対しては18,646 msから90 msへ、radec_pointに対しては924 msから36 msへと顕著に短くなった（ビューに対する検索についても数%以内で同じ程度の値）ことが効いている。非PostGISの場合には、キャッシュが有効になっても、テーブルに対する検索のBitmap Index Scanに要する時間が、crval1とcrval2の複合インデックスに対しては509 msから418 msへ、x,y,zの複合インデックスに対しては941 msから530 msへと変化するだけで、顕著な時間の短縮はみられない。Bitmap Index Scanにおいて、GEOGRAPHY型データに対しては、通常の型のデータよりも非常に強くキャッシュの効果が働くと考えられるが、その原因は不明である。

PostGIS(noST)の場合も実行時間が非PostGISの場合に比べテーブルでは84%程、ビューでは81%程短縮された。PostGIS(noST)の場合には検索範囲絞り込みの列としてcrval1とcrval2を用いており、それらに対するキャッシュの効果がGEOGRAPHY型データほどは強く効かないため、PostGIS(ST)の場合ほどは実行時間が短くはならなかった。

■ SMOKA/Tomo-e Gozen システムへの応用 SMOKA/Tomo-e Gozen システムの運用ではビューを検索対象としており、キャッシュ無効時には非PostGISの場合より実行時間が長くなるためPostGISの利用は適切でない。一方、キャッシュ有効時には非PostGISの場合に比べ90%前後短縮されるためPostGISの利用が薦められる。今後Tomo-e Gozenのデータ量が増大し、部分的にキャッシュが有効となる場合にPostGIS利用によって実行時間がどうなるかは不明であり、今後調査及び検討すべき課題である。

7 まとめ

本論文では、膨大な観測データ数を有する光学赤外線天文観測データアーカイブシステムのデータベースの検索速度を高速化するために、約2800万件のストック済みデータの情報を有するSMOKA/Tomo-e Gozenシステムのデータベースと、同システムが提供する検索機能であるピンポイント検索、ラフ検索、カレンダー検索のSQLクエリを使って、テーブル分割とパラレルクエリを併用した際の最適なテーブル分割数とパラレルクエリの設定値の調査（テーブル分割+パラレルクエリ実験）、テーブルファイルとインデックス

ファイルをSSD上に配置した場合の検索速度の調査（SSD実験）、PostGISによるR-tree空間インデックスを利用した場合の検索速度の調査（PostGIS実験）をおこなった。その結果、各技術ごとに検索速度を高速化するための設定とその効果が明らかになった。

テーブル分割+パラレルクエリ実験では、テーブル分割数の異なる9種類の実験用ビューを作成し、パラレルクエリに関するパラメータ設定値を4通りに変化させ、ビューに対してピンポイント検索、ラフ検索、カレンダー検索の実験用SQLを実行することで、最適なテーブル分割数とパラメータ設定値の組み合わせを調査した。結論は次のとおりである。

1. キャッシュ無効時、分割キーが問い合わせの検索条件に含まれる検索では、テーブル分割数を増やすことで実行時間を短縮できる。ただし、領域検索においては、テーブルの分割範囲が検索半径を下回ると被スキャン子テーブル数が急増するため、被スキャン子テーブル数とワーカー起動数のバランスから、分割範囲が検索範囲程度となるテーブル分割数が適切であると考えられる。
2. キャッシュ有効時のテーブル分割数の増加に伴う実行時間の変化量はキャッシュ無効時と比べ小さい。したがってテーブル分割数はキャッシュ無効時の実行時間が短いものを選ぶべきである。
3. パラレルクエリのパラメータ設定値の違いによる実行時間の系統的な変化はない。ただし、パラメータ設定値によりワーカー起動数が抑制されたと思われる例が見受けられたため、計算機資源に鑑み、パラメータ設定値は大きめの値にすべきと考える。

本実験環境においては、テーブル分割数が増えるにつれピンポイント検索とラフ検索のキャッシュ無効時の実行時間が減少した。非分割時と比べピンポイント検索は最大55%程、ラフ検索は最大83%程短縮された。

SSD実験では、テーブルファイルとインデックスファイルをSSD上に配置した場合、テーブル並びにビューに対する実験用SQLクエリの実行時間が、HDD上に配置した場合と比べどの程度高速化されるのか調査した。結論は次のとおりである。

1. SSDを利用することでキャッシュ無効時の実行時間を大きく短縮できる。
2. キャッシュ有効時の実行時間は短縮されない。

本実験環境においては、テーブルに対するキャッシュ無効時の実行時間がHDDを使用した場合と比べ、ピンポイント検索では最大97%程、ラフ検索では最大

91%程, カレンダー検索では最大98%程短縮された。ビューに対する場合, ピンポイント検索では最大98%程, ラフ検索では最大92%程, カレンダー検索では最大96%程短縮された。実行時間を大きく短縮できることから, 積極的にSSDを導入すべきと考える。

PostGISを利用した場合の検索速度の調査では, 検索テーブルの球面座標データをGISオブジェクトのGEOGRAPHY型POINTデータに変換したものにR-tree空間インデックスを追加し, 検索範囲の絞り込みにPostGIS関数を利用する場合に, テーブル並びにビューに対するピンポイント検索, ラフ検索の実験用SQLクエリの実行時間が, PostGISを利用しない場合と比べどの程度短縮されるのか調査した。結論は次のとおりである。

1. キャッシュが有効なデータベースではPostGISを利用することで領域検索の実行時間を短縮できる。
2. キャッシュが無効なデータベースでは逆に実行時間が長くなる。

本実験環境においては, テーブルに対するキャッシュ有効時の実行時間が非PostGISと比べ, ピンポイント検索では最大47%程, ラフ検索では最大90%程短縮された。ビューに対する場合, ピンポイント検索では最大23%程, ラフ検索では最大85%程短縮された。一方, キャッシュ無効時のテーブル並びにビューに対する実行時間は非PostGISと比べて長くなった。したがって計算機のメモリ容量とテーブルサイズを比較し, PostGIS導入の可否を決めるべきと考える。

膨大な観測データ数を有する光学赤外線天文観測データアーカイブシステムでは, その検索速度を高速化させる方法として, SSDへのテーブルファイルとインデックスファイルの配置が, 最も簡単かつ効果が期待できる。それに加えSMOKA/Tomo-e Gozenシステムでは, テーブル分割数を90分割に増やし, パラレルクエリの設定値をCPUコア数と同数の24程度に設定することで, さらなる高速化を図れると考えられる。PostGISを利用することでキャッシュ有効時のラフ検索の実行時間を大きく短縮できたことから, テーブル分割, パラレルクエリ, PostGISの併用することで, キャッシュ無効時と有効時の双方の実行時間を短縮できる可能性がある。

本論文のテーブル分割+パラレルクエリ実験では赤経方向にテーブルを分割したが, ピンポイント検索やラフ検索のような領域検索をおこなった際に, 極に近づくほど被スキャン子テーブル数が増えてしまう。極付近を検索した場合にどの程度実行時間が低下するのか検証が必要だろう。検証結果によっては赤緯方向へ

の分割に変更する必要があるかもしれない。

テーブル分割には本論文でおこなった方法以外にも, テーブルデータを列方向に管理する列指向テーブルを使って列ごとにテーブルを分割する方法や, テーブルデータを複数の計算機ノードに分散させる分散テーブル等の方法も存在する。前者は列方向のデータが似たような値を持つという特性を利用してデータを圧縮保存し, かつ検索に必要な列のみを読み込むことで検索時のデータ読み込み量を減らし, 検索速度の高速化を図るものである[36]。後者は複数の計算機ノードに分散させたテーブルデータに対し並列に検索処理を実行することで検索速度の高速化を図るものである[37]。PostgreSQLではCitrus [38]等の拡張機能を導入することで列指向テーブルや分散テーブルの機能を実装できるため, その性能を確かめてみたいところである。

PostGIS実験ではR-tree空間インデックスを使うためにPostGISを利用したが, 同様の拡張機能はPostGISの他にもPgSphere [39]やQ3C [40]がある。これらとの性能比較についても確かめてみたいところである。

謝辞

本論文の実験実施にあたって多大なる助言を天文データセンター市川伸一氏にいただいた。本論文の執筆に当たって多大なる助言を天文データセンター高田唯史氏と古澤久徳氏にいただいた。諸氏にはここで深く感謝する。匿名の査読者には有益なご指摘をいただいた。ここで深く感謝する。SMOKA/Tomo-e Gozenシステム及び本論文の実験用システムはその開発及び運用の大部分が国立天文台天文データセンターの経費によって賄われている。

補遺A ANALYZEの効果に関する調査

前論文[17]並びに本論文のテーブル分割+パラレルクエリ実験とSSD実験で使用したテーブルでは, テーブルへのデータのコピー後に手動でANALYZEコマンドを実行せず, 各種実験をおこなっていた。ANALYZEコマンドはテーブルの内容に関する統計情報を収集し, その結果をpg_statisticテーブルに格納するものである[41]。PostgreSQLのプランナは統計情報を元にコストの推計をおこない[42], コストが最小となる実行計画を生成する[26]。したがって統計情報の収集はデータベースの検索性能を保つ上で重要である。

PostgreSQL 12は, 不要タプルの回収をおこなうVACUUMコマンドと統計情報の更新をおこなうANALYZE

コマンド [41] を自動的に実行する、自動 Vacuum 機能を備えている [43]。自動 Vacuum は Vacuum 閾値並びに Analyze 閾値を超える INSERT, UPDATE, DELETE があるテーブルで発生した時に自動的に実行される。Analyze 閾値は $\text{autovacuum_analyze_threshold} + \text{autovacuum_analyze_scale_factor} \times \text{pg_class.reltuples}$ と定義され、PostgreSQL 12.7 では $\text{autovacuum_analyze_threshold}$ の初期値が 50、 $\text{autovacuum_analyze_scale_factor}$ の初期値が 0.1 に設定されている。例えば約 2800 万件の行数を持つテーブルの場合、 $50 + 0.1 \times 2800 \text{万} = 280 \text{万} 50 \text{行}$ を超えるレコードの INSERT, UPDATE, DELETE がおこなわれると、当該テーブルに対して自動的に ANALYZE コマンドが実行される。

前論文並びに本論文のテーブル分割+パラレルクエリ実験と SSD 実験で使用したテーブルは、テーブルへのデータのコピー後に手動で ANALYZE コマンドを実行せず、自動 Vacuum 機能によって実行される ANALYZE（以下、自動 ANALYZE）を頼りに、各種実験をおこなっていた。しかし実験をおこなう中で、自動 ANALYZE が実行されたテーブルと手動で ANALYZE コマンドを実行したテーブル、またテーブルの作成方法の違いで、SQL クエリの実行計画や実行時間に差が生じることに気づいた。本節では

ANALYZE コマンドの実行方法とテーブル作成方法の違いによる SQL クエリの実行計画や実行時間の変化についてまとめる。

A.1 実験方法

実験のため、第4節で使用した `tbl_fheader` と同じテーブル定義を持つ、`tbl_fheader_mcn`, `tbl_fheader_mca`, `tbl_fheader_min`, `tbl_fheader_lin` を作成した。これらのテーブルはテーブル定義は全く同じだが、テーブルの作成方法 (`m`: 列名とデータ型を定義して作成, `l`: LIKE 句によるコピー), テーブルへのデータのコピー方法 (`c`: COPY, `i`: INSERT), 手動 ANALYZE コマンドの有無 (`a`: 有, `n`: 無) が異なる。`tbl_fheader_mcn` では、CREATE TABLE コマンドで列名とデータ型を定義してテーブルを作成、CREATE INDEX コマンドでインデックスを作成、psql のメタコマンドである \COPY コマンドを使って観測日毎に 396 個のファイルに別れた FITS ファイルのヘッダー情報をテーブルにコピーした。`tbl_fheader_mca` では、`tbl_fheader_mcn` と同様の方法でテーブル・インデックスの作成とデータのコピーをおこなったのち、手動で ANALYZE コマンドを実行した。`tbl_fheader_min` では、`tbl_fheader_mcn` と同様の方法でテーブル・インデックスを作成したの

表 A1 `tbl_fheader_(mcn|mca|min|lin)` に対する実験用 SQL クエリの実行結果。実行時間の単位は ms、括弧内の値が標準誤差。スキャン方式の `I` は Index Scan を、`B` は Bitmap Heap Scan を表す。

		実行時間の平均値と標準誤差				スキャン方式	総読み出しページ数
		キャッシュ無効		キャッシュ有効			hit+read
ヒューマン ポ ジ ション	<code>tbl_fheader_mcn</code>	23443.5	(37.8)	80.7	(10.6)	I	4290
	<code>tbl_fheader_mca</code>	23221.1	(80.6)	66.8	(9.9)	I	4290
	<code>tbl_fheader_min</code>	23203.6	(68.2)	79.4	(10.1)	I	4290
	<code>tbl_fheader_lin</code>	29204.2	(14.4)	82.2	(10.8)	I	4473
マ シ ン	<code>tbl_fheader_mcn</code>	50054.1	(44.0)	5217.5	(7.2)	I	90354
	<code>tbl_fheader_mca</code>	21326.8	(532.1)	5335.3	(8.9)	B	73660
	<code>tbl_fheader_min</code>	21318.0	(428.8)	5405.7	(7.0)	B	73660
	<code>tbl_fheader_lin</code>	163069.1	(155.5)	5418.5	(10.3)	B	78245
カ レ ン ダ ー	<code>tbl_fheader_mcn</code>	862.5	(6.2)	200.8	(11.5)	I	11586
	<code>tbl_fheader_mca</code>	861.6	(0.3)	198.6	(11.7)	I	11583
	<code>tbl_fheader_min</code>	876.3	(17.8)	214.5	(15.1)	I	11583
	<code>tbl_fheader_lin</code>	863.0	(4.0)	203.9	(12.0)	I	11567

表 A2 `pg_stat_all_tables` より抽出した `tbl_fheader_(mcn|mca|min|lin)` の統計情報。

	<code>n_tup_ins</code>	<code>n_live_tup</code>	<code>n_mod_since_analyze</code>	<code>analyze_count</code>	<code>autoanalyze_conut</code>
<code>tbl_fheader_mcn</code>	27918958	27713601	1928139	0	15
<code>tbl_fheader_mca</code>	27918958	27924132	0	1	15
<code>tbl_fheader_min</code>	27918958	27916822	0	0	1
<code>tbl_fheader_lin</code>	27918958	27920555	0	0	1

ち、INSERT コマンドを使ってtbl_fheader_mcnからデータをコピーした。tbl_fheader_linでは、CREATE TABLE tablename (LIKE tbl_fheader_mcn INCLUDING ALL) コマンドでテーブル・インデックスを作成したのち、INSERT コマンドを使ってtbl_fheader_mcnからデータをコピーした。

作成したテーブルに対し、EXPLAIN (ANALYZE, BUFFERS)を冠したピンポイント検索、ラフ検索、カレンダー検索の実験用SQLを実行し、実行計画と読み出しページ数を調査した。またpg_stat_all_tablesテーブル[44]から統計情報の確認をおこなった。

A.2 実験結果

表A1は、tbl_fheader_(mcn |mca |min |lin)に対する実験用SQLクエリの実行結果である。実行時間の単位はms、括弧内の値が標準誤差である。スキャン方式と総読み出しページ数はキャッシュ無効時の実行計画から情報を読み取った。スキャン方式はテーブルに対して実行されたスキャンの方式であり、IがIndex Scanを、BがBitmap Heap Scanをそれぞれ表す。総読み出しページ数は問い合わせ処理全体で読み出されたページ数の総計であり、キャッシュ有効時における実行計画の最初に表示されるBuffersのhitとreadの合算値を掲載した。

表A2は、pg_stat_all_tablesより抽出したtbl_fheader_(mcn |mca |min |lin)の統計情報である。n_tup_insは挿入された行数、n_live_tupは有効行の推定値、n_mod_since_analyzeは最後にAnalyzeが実行されてから変更された行の推定値、analyze_countは手動でAnalyzeが実行された回数、autoanalyze_countは自動VacuumによってAnalyzeが実行された回数である[44]。

A.3 考察

表A1より、手動でのAnalyzeの有無並びにテーブルの作成方法の違いによって実行時間に差が生じることがわかった。

tbl_fheader_mcnとtbl_fheader_mcaを比べると、前者のラフ検索の実行時間が後者よりも2倍以上長くなった。実行計画を比較すると、前者ではIndex Scanが使われているのに対し後者ではBitmap Heap Scanが使われており、実行計画の違いが実行時間の差を生じさせたと考えられる。実行計画の違いは統計情報の違いによって生じたと考えられる。表A2より前者のn_mod_since_analyzeの値が1,928,139行であるのに対し後者は0行である。両テーブルともVACUUM コマンドによってテーブルの行数がAnalyze 閾値を越える毎に自動ANALYZEが実行され統計情報が更新された。しかし

約2800万件のデータの入力完了した時点で、約190万行が統計情報に反映されなかった。後者のテーブルでは手動ANALYZEによって統計情報が更新されたが前者はそのままであったため、実行計画が最適化されなかったのではないかと推測される。

tbl_fheader_minとtbl_fheader_linを比べると後者のピンポイント検索の実行時間が約1.3倍、ラフ検索の実行時間が約8倍、前者よりも長くなった。実行計画を比較すると、Unique処理終了時における前者と後者のCOSTの値がそれぞれ165129.90と183153.17、推測行数がそれぞれ71行と70行である等の違いがみられたが、使用された処理方法は同じであった。本実験で得られた情報からは実行時間に約8倍もの差が生じた原因はわからなかった。またtbl_fheader_linに対し手動でAnalyze コマンドを実行しても実行時間は改善しなかった。

以上の理由から、テーブルへのデータの追加や削除等をおこなった後は、統計情報を更新するために手動でAnalyze コマンドを実行すべきと考える。またLIKE INCLUDING ALLを使ったCREATE TABLEは、少なくともSMOKA/Tomo-e Gozen システムのデータベースでは利用すべきでないと考える。

参考文献

- [1] 馬場肇, 安田直樹, 市川伸一, 八木雅文, 岩本信之, 高田唯史, 洞口俊博, 多賀正敏, 渡邊大, 奥村真一郎, 小澤友彦, 山本直孝, 濱部勝: すばる望遠鏡公開データアーカイブシステムの開発, 国立天文台報, **6**, 23–26 (2002).
- [2] 山本直孝, 野田祥代, 多賀正敏, 小澤友彦, 洞口俊博, 奥村真一郎, 古荘玲子, 馬場肇, 八木雅文, 安田直樹, 高田唯史, 市川伸一: すばる望遠鏡公開データアーカイブシステムの開発2, 国立天文台報, **6**, 79–100 (2003).
- [3] 榎基宏, 多賀正敏, 小澤友彦, 野田祥代, 奥村真一郎, 吉野彰, 古荘玲子, 馬場肇, 洞口俊博, 高田唯史, 市川伸一: すばる望遠鏡公開データアーカイブシステムの開発3, 国立天文台報, **7**, 57–84 (2004).
- [4] 出田誠, 榎基宏, 小澤友彦, 吉野彰, 仲田史明, 奥村真一郎, 山本直孝, 古荘玲子, 矢治健太郎, 山田善彦, 八木雅文, 洞口俊博, 高田唯史, 市川伸一: すばる望遠鏡公開データアーカイブシステムの開発4, 国立天文台報, **8**, 59–84 (2005).

- [5] 山田善彦, 小澤友彦, 西澤淳, 古荘玲子, 西村高德, 榎基宏, 吉野彰, 古澤順子, 高田唯史, 市川伸一: すばる望遠鏡公開データアーカイブシステムの開発5, *国立天文台報*, **12**, 53–78 (2009).
- [6] 野田祥代, 古荘玲子, 古澤順子, 山田善彦, 山内千里, 小澤友彦, 高田唯史, 市川伸一: すばる望遠鏡公開データアーカイブシステムの開発6, *国立天文台報*, **14**, 35–61 (2012).
- [7] 中島康, 樋口あや, 格和純, 小野里宏樹, 野田祥代, 古澤順子, 本間英智, 高田唯史, 市川伸一: 光学赤外線観測データアーカイブシステム SMOKA: 20年間の開発と運用、そして将来, *国立天文台報*, **22**, 1–44 (2022).
- [8] 中島康, 小澤武揚, 小野里宏樹, 森由貴, 市川伸一: SMOKA/Tomo-e Gozen データ公開システムの開発, *国立天文台報*, **23**, 1–15 (2022).
- [9] The PostgreSQL Global Development Group: PostgreSQL ホーム ページ, 2023, <https://www.postgresql.org/>, (最終閲覧日: 2023-11-28).
- [10] Sako, S., et al.: The Tomo-e Gozen wide field CMOS camera for the Kiso Schmidt telescope, *Proc. SPIE*, **10702**, 107020J (2018).
- [11] Miyazaki, S., et al.: Hyper Suprime-Cam: System design and verification of image quality, *PASJ*, **70**, S1 (2018).
- [12] Komiyama, Y., et al.: Hyper Suprime-Cam: Camera dewar design, *PASJ*, **70**, S2 (2018).
- [13] Kawanomoto, S., et al.: Hyper Suprime-Cam: Filters, *PASJ*, **70**, 66 (2018).
- [14] Furusawa, H., et al.: The on-site quality-assurance system for Hyper Suprime-Cam: OSQAH, *PASJ*, **70**, S3 (2018).
- [15] PostgreSQL グローバル開発グループ: 「5.11. テーブルのパーティショニング」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/ddl-partitioning.html>, (最終閲覧日: 2023-11-28).
- [16] PostgreSQL グローバル開発グループ: 「第15 章パラレルクエリ」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/parallel-query.html>, (最終閲覧日: 2023-11-28).
- [17] 小澤武揚, 小野里宏樹, 中島康: 光学赤外線天文観測データアーカイブシステムにおける検索高速化の研究, *国立天文台報*, **23**, 16–44 (2022).
- [18] PostGIS ホームページ, 2023, <https://postgis.net/>, (最終閲覧日: 2023-11-28).
- [19] Guttman, A.: R-trees: a dynamic index structure for spatial searching, *ACM SIGMOD Record*, **14**(2), 47–57 (1984).
- [20] 板垣貴裕: 「パーティショニング: 用途と利点」, Let's Postgres, 2011, <https://lets.postgresql.jp/documents/technical/partitioning/1>, (最終閲覧日: 2023-11-28).
- [21] PostgreSQL グローバル開発グループ: 「68.6. データベースページのレイアウト」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/storage-page-layout.html>, (最終閲覧日: 2023-11-28).
- [22] PostgreSQL グローバル開発グループ: 「15.3. パラレルプラン」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/parallel-plans.html>, (最終閲覧日: 2023-11-29).
- [23] 株式会社インセプト: 「SSD」, IT 用語辞典 e-Words, 2022, <https://e-words.jp/w/SSD.html>, (最終閲覧日: 2023-11-29).
- [24] PRIMERGY Product Marketing, PRIMERGY Performance and Benchmarks: ホワイトペーパー Fujitsu PRIMERGY サーバソリッドステートドライブ- FAQ (v1.1), 2014, <https://sp.ts.fujitsu.com/dmsp/Publications/public/wp-solid-state-drives-faq-ww-ja.pdf>, (最終閲覧日: 2023-11-29).
- [25] Lehman, P., Yao, S.: Efficient locking for concurrent operations on B-trees, *ACM Trans. Database Systems*, **6**(4), 650–670 (1981).
- [26] PostgreSQL グローバル開発グループ: 「14.1. EXPLAIN の利用」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/using-explain.html>, (最終閲覧日: 2023-12-18).
- [27] PostgreSQL グローバル開発グループ: 「19.4. 資源の消費」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/runtime-config-resource.html>, (最終閲覧日: 2023-12-18).
- [28] PostgreSQL グローバル開発グループ: 「15.4. パラレル安全」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/parallel-safety.html>, (最終閲覧日: 2023-12-18).
- [29] PostgreSQL グローバル開発グループ: 「INSERT」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/sql-insert.html>, (最終閲覧日: 2023-12-18).
- [30] PostgreSQL グローバル開発グループ: 「EXPLAIN」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/sql-explain.html>, (最終閲覧日: 2023-12-18).
- [31] PostGIS PSC & OSGeo: 「4.3. ジオグラフィックデータタイプ」, PostGIS 3.3.6dev マニュアル, 2023, https://postgis.net/docs/manual-3.3/postgis-ja.html#PostGIS_Geography, (最終閲覧日: 2023-12-18).

- [32] PostGIS PSC & OSGeo: 「4.9.1. GiST インデックス」, PostGIS 3.3.6dev マニュアル, 2023, https://postgis.net/docs/manual-3.3/postgis-ja.html#gist_indexes, (最終閲覧日: 2023-12-18).
- [33] PostGIS PSC & OSGeo: 「ST MakeEnvelope」, PostGIS 3.3.6dev マニュアル, 2023, https://postgis.net/docs/manual-3.3/postgis-ja.html#ST_MakeEnvelope, (最終閲覧日: 2023-12-18).
- [34] Power, R. A.: Large Catalogue Query Performance in Relational Databases, *Publications of the Astronomical Society of Australia*, **24**, 13–20 (2007).
- [35] PostGIS PSC & OSGeo: 「ST DWithin」, PostGIS 3.3.6dev マニュアル, 2023, https://postgis.net/docs/manual-3.3/postgis-ja.html#ST_DWithin, (最終閲覧日: 2023-12-18).
- [36] 小森博之: Michael Stonebraker が生み出した列指向データベースは何が凄いのか? ~ Vertica を例に列指向データベースのアーキテクチャを詳解~, db teck showcase 2013 講演資料, 2013, <https://www.slideshare.net/InsightTechnology/d36-vertica-komori>, (最終閲覧日: 2023-12-18).
- [37] 篠田典良: Citus 10 検証結果, 日本ヒューレット・パカード合同会社, 2021, https://h50146.www5.hp.com/products/software/oe/linux/mainstream/support/lcc/pdf/citus10_20210902-1.pdf, (最終閲覧日: 2023-12-18).
- [38] Citus Data ホームページ, <https://www.citusdata.com/product>, (最終閲覧日: 2023-12-18).
- [39] PgSphere ホームページ, <https://pgsphere.github.io/>, (最終閲覧日: 2023-12-18).
- [40] Koposov, S., & Bartunov, O.: Q3C, Quad Tree Cube. The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL, Astronomical Data Analysis Software and Systems XV, *ASP Conference Series*, **351**, 735–738 (2006).
- [41] PostgreSQL グローバル開発グループ: 「SQL コマンド ANALYZE」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/sql-analyze.html>, (最終閲覧日: 2023-12-18).
- [42] PostgreSQL グローバル開発グループ: 「70.1. 行数推定の例」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/row-estimation-examples.html>, (最終閲覧日: 2023-12-18).
- [43] PostgreSQL グローバル開発グループ: 「24.1. 定常的なバキューム作業」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/routine-vacuuming.html>, (最終閲覧日: 2023-12-18).
- [44] PostgreSQL グローバル開発グループ: 「27.2. 統計情報コレクタ」, PostgreSQL 12.4 文書, 2020, <https://www.postgresql.jp/document/12/html/monitoring-stats.html>, (最終閲覧日: 2023-12-18).