国立天文台報　第 23 巻，16– 44（2022）　https://doi.org/10.50916/rnaoj.23.0_16

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

小澤武揚，小野里宏樹，中島 康

（2022 年 9 月 12 日受付；2022 年 11 月 22 日受理）

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

Takeaki OZAWA, Hiroki ONOZATO, Yasushi NAKAJIMA

Abstract

The number of data released by optical-infrared astronomical data archive systems has increased significantly in recent years with the advent of high-speed imaging instruments. In this paper, we investigate the effect of a huge number of data on the search speed and the database/table configuration to speed up searches in the optical-infrared astronomical observation data archive system. We conducted experiments on each search feature (pinpoint search, rough search, and calendar search) of the SMOKA/Tomo-e Gozen system, a data archive system for the Tomo-e Gozen high-speed camera. We report the results of these experiments and discuss the optimal database/table configurations for the optical-infrared astronomical data archive system with a huge number of data.

概要

高速撮像観測装置の登場により，光学赤外線天文観測データアーカイブシステムが公開するデータ数は近年著しく増加している．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におけるデータ数の巨大化が検索速度に与える影響及び，検索速度の高速化を実現するためのデータベースの設定とデータベーステーブルの構成について調査した．高速撮像観測装置 Tomo-e Gozen のデータアーカイブシステムである SMOKA/Tomo-e Gozen システムを調査の対象とし，当システムの各検索機能（ピンポイント検索，ラフ検索，カレンダー検索）について実験をおこなった．これらの実験結果を報告し，データ数が巨大な光学赤外線天文観測データアーカイブシステムの最適なデータベースの設定等について論ずる．

1　はじめに

天文観測データはある時刻ある方角の状態を捉えた再現不可能なデータであり，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であると言える．国立天文台天文データセンターではSMOKAシステム[1-7]やSMOKA/Tomo-e Gozenシステム[8]等を運用して観測データの公開をおこなっている．

天文観測データアーカイブシステムの運用において利用者が必要とする観測データを迅速かつ正確に見つけるためには，観測データのメタデータ（観測日時や赤経・赤緯など）を管理しその検索機能を提供するデータベース管理システム（以下DBMS）が必須である．天文データセンターではDBMSとしてPostgreSQLを用いて検索機能を提供している．

近年の観測装置の進化は，観測データのFITSファイルの構造の複雑化や，生成される観測データの数と量の巨大化を招いている[7]．巨大な観測データが生成されている観測装置の例として，東京大学木曽観測所105 cmシュミット望遠鏡のTomo-e Gozen [9]や京都大学岡山天文台せいめい望遠鏡のTriCCS [10]に代表される高速撮像観測装置が挙げられる．観測データ数の巨大化は，データベースのテーブルファイルやインデックスファイルの巨大化，検索ヒット数の増加とディスクアクセス量の増大，メモリ使用量の圧迫等，データベースの検索時間を増大させるであろう様々な問題を招くと考えられる．

主な検索対象キーが観測日時及び天球座標であることが天文観測データアーカイブシステムの特徴である．そのような特徴を持つデータアーカイブシステムにおいて，観測データ数の巨大化が検索時間に与える影響を定量的に調べること，そして検索時間を短縮化させる方法を知ることは天文観測データアーカイブシステムの開発と運用にとって重要な課題である．

観測データ数が巨大化したアーカイブシステムのデータベースを高速に動作させる方法としては，より性能の高い計算機や商用の高機能DBMSを導入する等の方法が考えられる．しかし諸般の事情によりそれらの導入は難しいことが多い．我々は，高性能計算機や商用の高機能DBMSを導入するのではなく，既存の計算機及びDBMSを使用してデータベースの設定の改善で本課題を克服できるのか調査をおこなった．調査対象として，SMOKA/Tomo-e Gozenシステムのデータベースを選んだ．

SMOKA/Tomo-e Gozenシステムは木曽観測所シュミット望遠鏡の可視光広視野高速撮像装置Tomo-e Gozenの観測データを公開するためのシステムである．同システムではTomo-e Gozenで得られた観測データのうち，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[1) 2)]の観測データに一次処理（バイアス，ダーク，フラット較正），位置較正及びスタック処理を施したデータ（以降，スタック済みデータ）を公開している．公開中のデータのフレーム数（本論文ではフレームとは一つのFITSファイルを指すものとする）は2022年10月現在，9,153,756フレームであるが，未公開のデータを含めると本システムが保有するフレーム数は27,918,959フレームにのぼる．これは34観測装置を扱うSMOKAシステムが過去20年間に公開した約3500万フレームに匹敵する量であり[7]，同時に我々がデータを公開している観測装置の中で装置あたりのフレーム数が最大のものでもある．単一の観測装置のデータであることからSQLクエリが単純化され実験の分析が容易になるであろうと期待されることと，人工的なデータの偏りが発生しうるシミュレーションデータではなく実際の観測データであることから本観測装置のデータを用いることにした．

本論文の2章では本論文の調査対象であるSMOKA/Tomo-e Gozenシステムのデータベースの概要について記述し，3章ではSMOKA/Tomo-e Gozenシステムで使われている検索について総データ量と検索時間の間にどのような関係があるのか調査する．4章ではSMOKA/Tomo-e Gozenシステムで使われている検索の高速化について調査し，5章では本論文のまとめと今後に向けての課題について論ずる．

2　SMOKA/Tomo-e Gozenシステムのデータベースの概要

SMOKA/Tomo-e Gozenシステムのウェブインターフェースにはピンポイント検索，ラフ検索，カレンダー検索という3種類の検索機能が用意されている[8]．ピンポイント検索は指定した天球座標を画像内に含むフレームを検索するものであり，各フレームが位置較正済みであることを前提としている．ユーザが指定した天球座標から46分角（フレーム視野対角）以内にFITSヘッダのCRVAL1とCRVAL2の値が含まれるフレームを候補として絞り込む．候補となったそれぞれのフレームについて指定した天球座標が画像内にあるか位置較正情報を用いて判断し，該当するフレームの一覧をウェブページに表示する．ラフ検索は指定した

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

天球座標を「おおむね」含むショットを検索するものである．Tomo-e Gozenは84個のCMOS検出器から成るモザイクカメラであり，ショットとは同じ時刻に得られた最大84枚のフレームの組のことを指す．ラフ検索では位置較正済みの座標値を用いず，ユーザが指定した天球座標から半径6度以内にFITSヘッダのRAとDEC（望遠鏡のポインティング座標であり，ショットの視野範囲のほぼ中心を指す）が含まれるショットを検索し，その一覧をウェブページに表示する．カレンダー検索は指定した観測日に取得されたフレームを検索するものである．ユーザーが指定した観測日とFITSヘッダのDATE-OBSの値が一致するフレームを検索し，その一覧をウェブページに表示する．なおピンポイント検索とラフ検索は観測日による絞り込み検索機能も持つが，本論文ではその機能は使用しない．

ウェブインターフェース用プログラムはJAVAで開発しており，スタック済みデータの情報を格納したデータベースとの連携にはJDBCを使っている．同プログラムが動作するウェブサーバとDBMSが動作するデータベースサーバは同一の計算機上で稼働しており，その諸元は表1の通りである．ウェブサーバの構築にはApache HTTP ServerとApache Tomcatを使い，データベースサーバの構築にはPostgreSQL 12.7を使っている．検索速度向上のためデータベースサーバの制御ファイルであるpostgresql.confに表2の変更を加えている．

SMOKA/Tomo-e Gozenシステムのデータベースにはスタック済みデータのFITSヘッダの全キーワードの値を管理するtmq_kisoテーブル，スタック済みデータの公開日や公開の可否等を管理するfilemng_tmqテーブル，公開対象かつ公開日を迎えた行を両テーブルから参照するtmq_smokaビューが存在する．tmq_smokaビューはラフ検索，ピンポイント検索，カレンダー検索の問い合わせ先となっており，検索に必要かつウェブインターフェースでユーザに提供すべき列を両テーブルから参照している[8，図4参照]．

SQL 1，SQL 2，SQL 3（補遺A）はそれぞれtmq_kisoテーブル，filemng_tmqテーブル，tmq_smokaビューの定義である．SQL 1では簡略化のためtmq_smokaビューから参照されるtmq_kisoテーブルの列のみを記載している．

2022年10月現在，tmq_kisoテーブルは行数が9,153,756行，列数が154列，テーブルファイルの大きさが~12.50 GBであり，1行あたり平均~1,365 Byteのデータ量を持つ．スタック済みデータは一部例外を除きFITSヘッダのFRAME_ID [3)]で識別可能であることからframe_id列に主キー制約を課し，tmq_smokaビューから参照されかつNULL値が存在してはならない列にNOT NULL制約を課している．またtmq_smoka

3）観測日付や露出の通し番号，検出器の番号などの文字列から成るフレームの識別子[8].

表1　SMOKA/Tomo-e Gozen計算機諸元.

ウェブサーバ兼データベースサーバ	CPU	Intel Xeon E5-2667 v4 3.2 GHz 8 core × 2
	RAM	DDR4 2400 RDIMM 8 GB × 8
	OS	Red Hat Enterprise Linux Server release 7.9
データベースサーバ用ディスク装置	実効容量	12 TB
	DISK	1.8 TB 2.5 inch SAS 10 krpm
	RAID	RAID6 (8 + 2)

表2　SMOKA/Tomo-e Gozenシステムの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

ビューから参照され検索条件で使われる列にはB-treeインデックス[11]を設定している．赤経と赤緯など必ず組となって検索がおこなわれる列には複合インデックス[12]を採用し，検索時間の短縮化を図っている．tmq_kiso_xyz_idxは，天球座標のra列とdec列の値からそれらと等価な直交座標系のx列，y列，z列の値を計算する関数feq2x，feq2y，feq2zから成る式インデックス[13]である．本式インデックスを定義することで，tmq_smokaビューの検索の高速化を図っている．

filemng_tmqテーブルは行数が9,153,756行，列数が6列，テーブルファイルの大きさが~3.88 GBであり，1行あたり平均~139 Byteのデータ量を持つ．必要な文字列長を設定したCHARACTER VARYING型を各列に設定している．tmq_kisoテーブルと同じ理由でframe_id列とdate_obs列（観測日）にそれぞれ主キー制約とNOT NULL制約を課している．またtmq_smokaビューの定義で検索条件となっている，データの公開日が格納されるpublictime列とデータ公開の可否の情報が格納されるpublicflag列にはB-treeインデックスを設定している．

tmq_smokaビューは，tmq_kisoテーブルから検索に使用される列を，filemng_tmqテーブルからpublictime列及びpublicflag列をそれぞれ参照し，加えて上記の関数feq2x，feq2y，feq2zを使って新たにx列，y列，z列を定義している．feq2x，feq2y，feq2zはra列とdec列を引数とした関数である．列に対する演算結果にはインデックスが適用されないため，ra列とdec列にインデックスが設定されていたとしても，戻り値であるx列，y列，z列にはインデックスが存在しない．したがってx列，y列，z列を検索条件とする検索ではIndex Scan [14]がおこなわれず検索時間が長くなってしまう．これを避けるためtmq_kisoテーブルに前述した式インデックスtmq_kiso_xyz_idxを作成し，x列，y列，z列を検索条件とした検索であってもIndex Scanがおこなわれるようにした．また本ビューではpublictime列とpublicflag列に対して検索条件を課すことで，公開対象かつ公開日に達したスタック済みデータのみを自動的にラフ検索，ピンポイント検索，カレンダー検索の検索対象としている．

SQL 4，SQL 5，SQL 6（補遺A）はピンポイント検索，ラフ検索，カレンダー検索のSQLクエリである．各SQLクエリのWHERE句内並びにラフ検索の関数fconesearchtmqsmoka（補遺A SQL 7）の引数に検索条件（観測日や赤経・赤緯）を指定する（SQL 4，SQL 5，SQL 6では3章の実験で使用した検索条件の値が入力されている）．ピンポイント検索とカレンダー検索がtmq_smokaビューを問い合わせ先とする一方，ラフ検索のSQLクエリはtmq_smokaビューを使ったユーザ定義関数であるfconesearchtmqsmokaを問い合わせ先としている．このユーザ定義関数は，検索中心と被検索座標の位置ベクトルのなす角を求め，求めた角度が任意角度（この例では6度）以下となる被検索座標を抽出する，いわゆるCone Search [15]を実施するものである．これらのSQLクエリではtmq_kisoテーブル及びfilemng_tmqに設定されたインデックスを利用したIndex Scanによる検索がおこなわれる．

3　観測データ数の増加が検索速度に与える影響

観測データ数の増加がデータベースの検索速度に与える影響を知るため，SMOKA/Tomo-e Gozenシステムのtmq_kisoテーブルとfilemng_tmqテーブルのデータ数を増加させていった際に，tmq_kisoテーブル及びtmq_smokaビューに対する検索時間がどのように変化するのか調査した．可能な限り多くのデータを使用するため，同システムが有する未公開分も含む約2800万フレームのスタック済みデータを使用した．検索クエリとして同システムのラフ検索，ピンポイント検索，カレンダー検索のSQLクエリを使用した．運用に影響を及ぼさないよう，実験は運用中のSMOKA/Tomo-e Gozenシステムから独立した実験用システム上でおこなった．

3.1　実験用システム

実験用データ

実験に使用したデータは観測日が2019-10-01から2021-05-14までの27,918,959フレームのスタック済みデータから，解析処理が複数回おこなわれファイル名が重複する1組2フレームのうち日付の古い1フレームを除いた27,918,958フレームである．図1左は，実験用データのショットの中心座標の天球面上での分布である．おおむね決まった格子点をショットの中心とした観測がおこなわれている．格子点は木曽観測所から観測可能な天球面上で一様に分布しているが，観測回数にムラがあることがわかる．図1右は各観測日に取得されたスタック済みデータの数の分布である．ほぼ一様に分布するが，ところどころに観測がおこなわれなかった期間がみられる．

実験用計算機

表3は実験に使用した計算機の諸元である．既存の計算機を流用したためSMOKA/Tomo-e Gozenシステムの計算機とは性能が異なる．SMOKA/Tomo-e Gozen

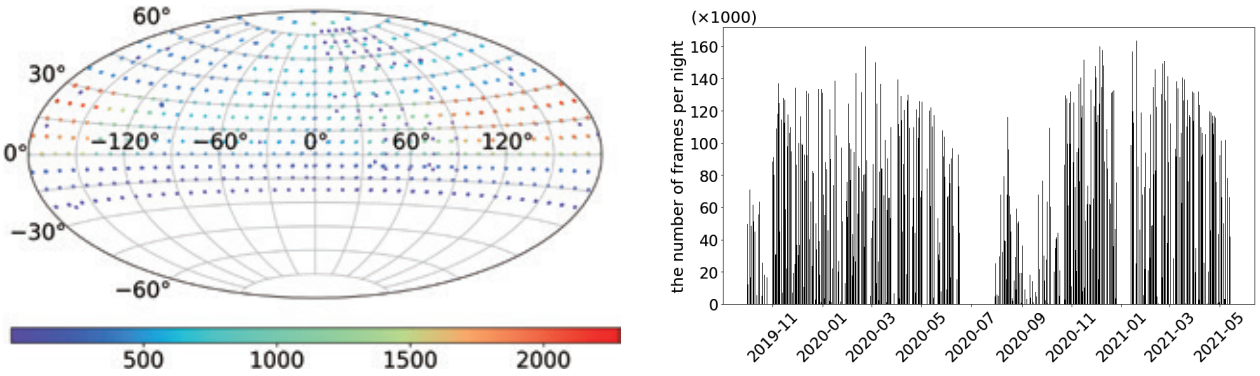

図1　左：実験用データのショットの中心座標の天球面上の分布．各点の半径1度以内にその中心座標が含まれるショットの数をカラーバーで示した色で表している．右：各観測日に取得されたスタック済みデータの数の分布．

表3　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	8 TB 3.5 inch SATA 6.0 Gbps 7200 rpm

システムの計算機と実験機の性能を比較すると，CPUコア数は前者が16コアであるのに対し後者は24コア，RAMの容量は前者が64 GBであるのに対し後者は128 GBである．コア数，RAMの容量とも実験機と異なるが，同性能の計算機を用意することは困難であった．本実験の目的はSMOKA/Tomo-e Gozenシステムの計算機の絶対的な性能を知ることではなくデータ増加時の検索時間の変化を知ることが目的であるため，性能が揃っていなくとも本論文の結論に影響はないと考えた．実験用計算機には3.5 inchのSATA接続のディスクを8本収容可能であり，システム領域用に1本，DBMSのテーブル空間用に7本のHDD（識別名：hdd1，hdd2，hdd3，hdd4，hdd5，hdd6，hdd7）を接続した．

DBMSはSMOKA/Tomo-e Gozenシステムと同じPostgreSQL 12.7を使用した．データベースサーバの制御ファイルであるpostgresql.confもSMOKA/Tomo-e Gozenシステムと同じ設定とした（表2参照）．

3.2　実験内容

SMOKA/Tomo-e GozenシステムのSQLクエリの問い合わせ先はビューであるが，一般に天文観測データアーカイブシステムにおいて問い合わせ先はビューであるとは限らない．例えばSMOKAシステムではテーブルが問い合わせ先である[7]．SMOKA/Tomo-e Gozenシステムにおける検索の高速化のみならず一般の天文観測データアーカイブシステムの高速化についても議論するため，問い合わせ先がビューとテーブルの場合の両方で実験をおこなった．なおマテリアライズドビュー[16]も問い合わせ先となり得るがその場合の実験はおこなわなかった．マテリアライズドビューを問い合わせ先とする調査は今後の課題としたい．

実験には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クエリの問い合わせ先を，後述する実験用テーブル及びビューに置き換えたものを使用した．ピンポイント検索では，fconesearchtmqsmoka関数（SQL 7）のFROM句のtmq_smokaビューを実験用テーブル及びビューに置き換えた関数を作成し，ピンポイント検索のSQLクエリ（SQL 4）のFROM句のfconesearchtmqsmoka関数を作成した関数に置き換えた．ラフ検索（SQL 5）とカレンダー検索（SQL 6）では，FROM句のtmq_smokaビューをそれぞれ実験用テーブル及びビューに置き換えた．全ての実験でSQLクエリの検索条件を固定し，ピンポイント検索では赤経・赤緯が$(00^h42^m44.32^s, +41°16'07.5'')$，ラフ検索では赤経・赤緯が$(05^h35^m17.29^s, -05°23'27.9'')$，カレンダー検索では観測日が2020-01-02である行を検索した．SQL 4，SQL 5，SQL 6ではこれらの値が検索条件として入力されている．

表4は使用した実験用テーブルとビューの一覧である．tbl_fheader*及びview_tomoe*を，各SQLクエリの

問い合わせ先とした．tbl_fheader* と tbl_filemng*，並びに view_tomoe* は SMOKA/Tomo-e Gozen システムの tmq_kiso テーブル（SQL 1）と filemng_tmq テーブル（SQL 2），並びに tmq_smoka ビュー（SQL 3）と同じ構造を持つテーブルとビューである．ただし view_tomoe* では WHERE 句内の CURRENT_TIMESTAMP を '2030-01-01 00:00:00' に変更して，全ての行が SQL クエリの検索対象となるようにした．tbl_fheader* と tbl_filemng* には表4のフレーム数列に示した数のスタック済みデータの情報を格納した．tbl_fheader と tbl_filemng 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に入力したのち，両テーブルから各テーブルへ当該行数のデータを frame_id 昇順に入力した．テーブルファイルとインデックスファイルの配置場所はディスク hdd1上である．約2800万フレームのスタック済みデータの情報を入力したテーブルの各ファイルの容量は表5の通りである．各ファイルの容量は psql のメタコマンドである \d+ を使って調べた．

検索時間の測定は PostgreSQL の EXPLAIN ANALYZE コマンド [17] を使っておこなった．同コマンドを冠した SQL クエリを実行して PostgreSQL のプランナ [18]（以降，プランナ）が作成した実行計画 [17]（以降，実行計画）を出力し，そこに表示される Execution time の値を SQL クエリの実行時間（以降，実行時間）とした．キャッシュ無効時及び有効時の双方について測定値を求めた．キャッシュ無効時及び有効時とは，計算機のディスクキャッシュと PostgreSQL の共有キャッシュに当該 SQL クエリの対象のテーブルのデータがキャッシュされていない状態とキャッシュされている状態のそれぞれを指す．キャッシュの削除は計算機を再起動することでおこなった．計算機を再起動した直後をキャッシュ無効時とし，2回目以降の当該 SQL クエリ実行時をキャッシュ有効時とした．キャッシュ無効時は1回の測定値を測定結果とし

表4　実験用テーブルとビューの一覧.

テーブル	テーブル	ビュー	フレーム数
tbl_fheader_50k	tbl_filemng_50k	view_tomoe_50k	50,000
tbl_fheader_100k	tbl_filemng_100k	view_tomoe_100k	100,000
tbl_fheader_200k	tbl_filemng_200k	view_tomoe_200k	200,000
tbl_fheader_400k	tbl_filemng_400k	view_tomoe_400k	400,000
tbl_fheader_800k	tbl_filemng_800k	view_tomoe_800k	800,000
tbl_fheader_1600k	tbl_filemng_1600k	view_tomoe_1600k	1,600,000
tbl_fheader_3200k	tbl_filemng_3200k	view_tomoe_3200k	3,200,000
tbl_fheader_6400k	tbl_filemng_6400k	view_tomoe_6400k	6,400,000
tbl_fheader_12800k	tbl_filemng_12800k	view_tomoe_12800k	12,800,000
tbl_fheader	tbl_filemng	view_tomoe	27,918,958

表5　約2800万フレームのスタック済みデータの情報が入力されたテーブルのテーブルファイルとインデックスファイルの容量.

ファイル名	ファイルの内容	容量（GiB）
tbl_fheader	テーブル	~38.2
tbl_fheader_pkey	主キー（frame_id）	~1.4
tbl_fheader_date_obs_idx	インデックス（date_obs）	~0.9
tbl_fheader_exp_id_idx	インデックス（exp_id）	~0.6
tbl_fheader_crpix1_crpix2_idx	インデックス（crpix1,crpix2）	~1.1
tbl_fheader_crval1_crval2_idx	インデックス（crval1,crval2）	~1.1
tbl_fheader_cd1_1_cd1_2_cd2_1_cd2_2_idx	インデックス（cd1_1,cd1_2,cd2_1,cd2_2）	~1.8
tbl_fheader_xyz_idx	インデックス（feq2x,feq2y,feq2z）	~1.6
tbl_filemng	テーブル	~3.8
tbl_filemng_pkey	主キー（frame_id）	~1.1
tbl_filemng_publictime_idx	インデックス（publictime）	~0.9
tbl_filemng_publicflag_idx	インデックス（publicflag）	~0.6

て，キャッシュ有効時は 4回の測定の平均値を測定結果とした．なお，キャッシュの削除はシェルコマンドでPostgreSQLサービスの停止，ディスクキャッシュのクリア，PostgreSQLサービスの起動をおこなうことでも可能である．しかし4節でおこなった実験において，SQLクエリの実行時間が再起動実行時よりもシェルコマンド使用時の方が有意に短い場合があった．シェルコマンド使用時に実行時間が短くなった原因が不明であり，キャッシュを完全に削除できていない疑いもあったため，本論文では再起動を採用した．

キャッシュ有効時と無効時の双方の実行時間を測定した理由は，キャッシュが無効な場合と比較してキャッシュが有効の場合に実行時間がどの程度短くなるのかを調べるためである．キャッシュ有効時に実行時間が短くなること自体は自明なことではあるが，その効果が定量的にどの程度であるのかは自明ではない．

また，SMOKA/Tomo-e Gozenシステムではウェブページに1度に表示される検索結果は100件である．表示に必要な件数のみを検索で出力することで効率の良い検索ができる可能性がある．一度に出力される検索結果数に制限をかけた場合に実行時間が変化するかどうかについても調べた．SQLクエリの末尾にLIMIT 100を付加することで検索結果数を100件に制限した．

3.3 実験結果

ピンポイント検索のデータ総数と実行時間との関係を表6と図2に，ラフ検索のデータ総数と実行時間との関係を表7と図3に，カレンダー検索のデータ総数と実行時間との関係を表8と図4に示す．ラフ検索とカレンダー検索ではSQLクエリにDISTINCT句が含まれており，検索処理の最終段階で重複行が削除される．検索処理数と実行時間との関係をみるために，重複行が削除される前の検索ヒット数を表7と表8に併記した．

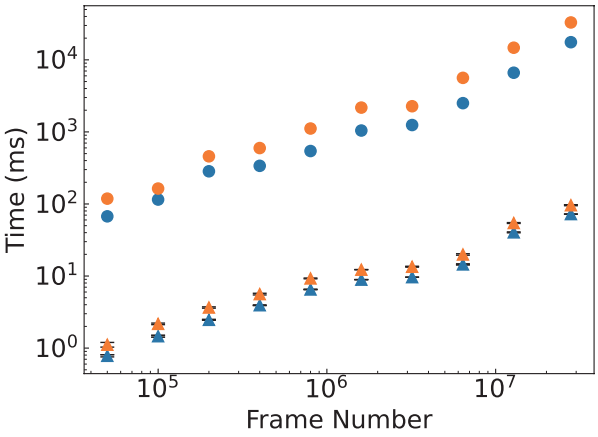

図2　ピンポイント検索のデータ総数と実行時間．青はテーブルを対象とした測定結果であり，橙はビューを対象とした測定結果である．それぞれ，丸はキャッシュが無効な場合，三角はキャッシュが有効な場合の測定結果である．キャッシュ有効時は4回の測定をおこないその平均を測定結果とし，その標準偏差を誤差棒で示した．

キャッシュ無効時と有効時の実行時間を比較すると，無効時は有効時と比べ0.5–2桁程度実行時間が長くなった．テーブルとビューに対する実行時間を比較すると，検索ヒット数が0件の場合や実行計画が変化した場合を除き，ビューはテーブルに比べ1.5–2倍程度実行時間が長くなった．

ピンポイント検索ではデータ総数と実行時間との間に正の相関がみられた（図2）．検索ヒット数と実行時間との関係（図5）においても正の相関がみられ，特にキャッシュ有効時はほぼ比例関係であった．キャッシュ有効時はデータ総数と実行時間との関係よりも，検索ヒット数と実行時間との関係の方がデータのばら

表6　ピンポイント検索のデータ総数と実行時間．実行時間の単位はmsである．

データ総数	テーブル			ビュー			検索ヒット数
	キャッシュ		スキャン方式	キャッシュ		結合方式	
	無効	有効		無効	有効		
50,000	67.4	0.8	Bitmap Index Scan	118.6	1.1	Nested Loop	39
100,000	115.2	1.5	Bitmap Index Scan	163.7	2.2	Nested Loop	81
200,000	284.1	2.5	Index Scan	458.2	3.7	Nested Loop	132
400,000	338.3	3.9	Bitmap Index Scan	596.9	5.6	Nested Loop	183
800,000	542.8	6.5	Bitmap Index Scan	1114.2	9.3	Nested Loop	306
1,600,000	1044.9	8.9	Index Scan	2174.7	12.3	Nested Loop	408
3,200,000	1245.6	9.7	Index Scan	2271.3	13.5	Nested Loop	444
6,400,000	2505.7	14.5	Index Scan	5624.5	19.9	Nested Loop	640
12,800,000	6626.7	40.4	Index Scan	14736.5	54.4	Nested Loop	1892
27,918,958	17572.1	72.4	Index Scan	33021.8	96.1	Nested Loop	3676

表7　ラフ検索のデータ総数と実行時間．実行時間の単位はmsである．

データ総数	テーブル キャッシュ 無効	テーブル キャッシュ 有効	テーブル スキャン方式	ビュー キャッシュ 無効	ビュー キャッシュ 有効	結合方式	検索ヒット数（重複行削除前）
50,000	28.7	0.5	Index Scan	13.9	0.5	Nested Loop	0 （0）
100,000	34.5	0.5	Index Scan	18.2	0.5	Nested Loop	0 （0）
200,000	180.0	12.3	Index Scan	232.8	12.5	Nested Loop	3 （21）
400,000	531.8	94.9	Bitmap Index Scan	665.0	94.7	Nested Loop	26 （294）
800,000	1229.0	180.2	Bitmap Index Scan	1642.1	185.1	Nested Loop	102 （1171）
1,600,000	3258.7	476.3	Bitmap Index Scan	4013.7	496.2	Nested Loop	418 （4709）
3,200,000	5252.8	692.3	Bitmap Index Scan	7025.7	724.2	Nested Loop	619 （7459）
6,400,000	19992.4	1138.9	Bitmap Index Scan	24176.4	1178.7	Nested Loop	1346 （14446）
12,800,000	53310.7	1982.3	Bitmap Index Scan	60907.8	2081.6	Nested Loop	1360 （27106）
27,918,958	39207.1	5060.0	Index Scan	79705.2	5325.6	Nested Loop	1366 （80881）

表8　カレンダー検索のデータ総数と実行時間．実行時間の単位はmsである．

データ総数	テーブル キャッシュ 無効	テーブル キャッシュ 有効	テーブル スキャン方式	ビュー キャッシュ 無効	ビュー キャッシュ 有効	結合方式	検索ヒット数（重複行削除前）
50,000	29.4	0.1	Index Scan	14.4	0.1	Nested Loop	0 （0）
100,000	12.7	0.1	Index Scan	12.8	0.1	Nested Loop	0 （0）
200,000	21.6	0.1	Index Scan	21.8	0.1	Nested Loop	0 （0）
400,000	36.6	0.1	Index Scan	18.8	0.1	Nested Loop	0 （0）
800,000	17.9	0.1	Index Scan	17.3	0.1	Nested Loop	0 （0）
1,600,000	159.5	62.1	Index Scan	1368.0	1131.4	Hash Join	899 （17426）
3,200,000	152.8	62.5	Index Scan	219.7	131.9	Nested Loop	899 （17426）
6,400,000	158.2	62.4	Index Scan	222.3	132.6	Nested Loop	899 （17426）
12,800,000	331.6	124.7	Index Scan	504.7	246.0	Nested Loop	907 （34761）
27,918,958	602.8	187.3	Index Scan	946.1	449.6	Nested Loop	911 （67906）

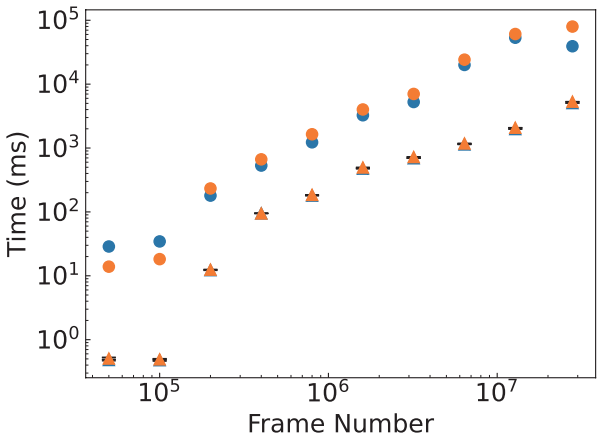

図3　ラフ検索のデータ総数と実行時間．測定点の表示方法（印の種類と色）は図2と同様である．

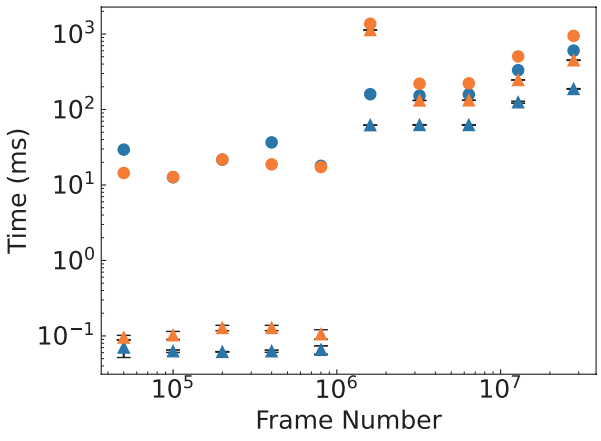

図4　カレンダー検索のデータ総数と実行時間．測定点の表示方法（印の種類と色）は図2と同様である．

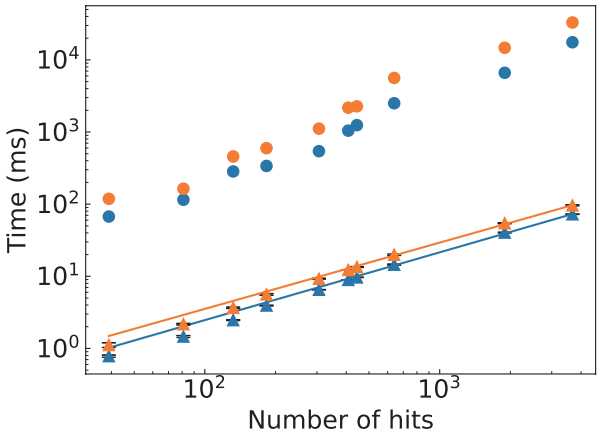

図5　ピンポイント検索の検索ヒット数と実行時間．測定点の表示方法（印の種類と色）は図2と同様である．青と橙の直線はキャッシュ有効時のテーブルの測定結果に対する回帰直線（$y = 0.033x^{0.939}$）とビューの測定結果に対する回帰直線（$y = 0.051x^{0.919}$）を表す．

つきが小さく，検索ヒット数の増加が実行時間の増加に影響を及ぼしているものと考えられる．

ラフ検索でもデータ総数と実行時間との間に正の相関がみられたが，データ総数が10^5フレーム以下と10^7フレーム以上のテーブルではその傾向が異なる（図3）．次の理由によりデータ総数と実行時間との関係が変化したと考えられる．データ総数が10^5フレーム以下のテーブルでは，検索条件に一致するデータが0件であった．Index Scanでは検索条件に一致するデータのみをテーブルファイルから抽出するため，一致するデータがない場合はテーブルファイルの読み込みがおこなわれない．そのためデータ総数と実行時間との間に正の相関が現れなかったものと考えられる．一方データ総数が10^7以上のテーブルでは，SQLクエリの実行計画が変化したことがその原因であると考えられる．データ総数が400,000フレームから12,800,000フレームまでのテーブルに対するSQLクエリの実行計画ではBitmap Index Scan [19]がスキャン方式として使用されているが，27,918,958フレームのテーブルに対するSQLクエリではIndex Scanが使用されている．PostgreSQLのプランナは実行計画の優劣を比較するためにコスト[17]と呼ばれる値を使用するが，データ総数が27,918,958フレームの時はBitmap Index ScanよりもIndex Scanを使った実行計画の方がコストが小さいと判断したものと思われる．図6は重複行が削除される前の検索ヒット数と実行時間との関係である．キャッシュ有効時は両対数グラフ上での回帰直線からのデータのばらつきが小さく，ラフ検索においても，検索ヒット数の増加が実行時間の増加に影響を与えていると考えられる．

カレンダー検索ではデータ総数が10^7フレーム以下のテーブルでデータ総数と実行時間との間に正の相関がみられなかった（図4）．データ総数が10^6フレーム以下のテーブルでは検索ヒット数が0件であったため，その実行時間がほとんど変わらなかったものと考えられる．データ総数が10^6フレームから10^7までのテーブルでは検索ヒット数が899件で一定であったため，その実行時間がほとんど変わらなかったものと考えられる．データ総数が1,600,000フレームのビューの実行時間が外れ値となった理由は，実行計画の結合方式がNested Loop [20]からHash Scan [20]に変更されたためと考えられる．実行計画が変更された理由は不明である．データ総数が10^7フレーム以上のテーブルで

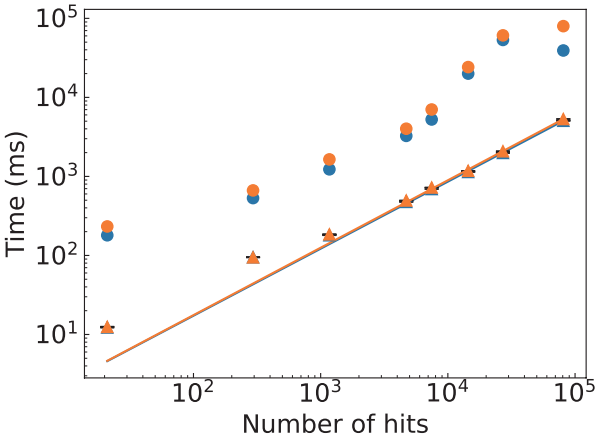

図6　ラフ検索のDISTINCT前の検索ヒット数と実行時間．測定点の表示方法（印の種類と色）は図2と同様である．対数プロットのため，検索ヒット数が0の実行時間は除いている．青と橙の直線はキャッシュ有効時のテーブルの測定結果に対する回帰直線（$y = 0.348x^{0.848}$）とビューの測定結果に対する回帰直線（$y = 0.346x^{0.853}$）を表す．

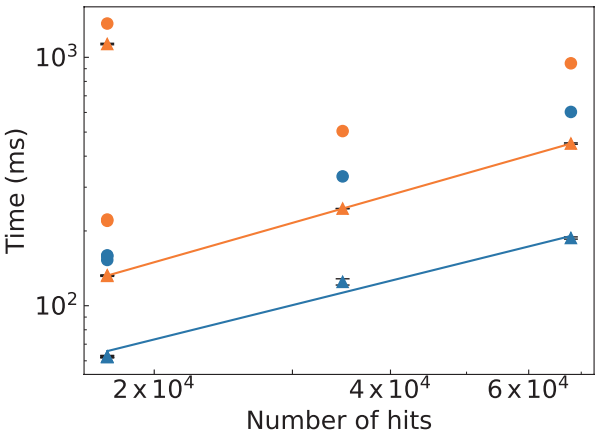

図7　カレンダー検索のDISTINCT前の検索ヒット数と実行時間．測定点の表示方法（印の種類と色）は図2と同様である．対数プロットのため，検索ヒット数が0の実行時間は除いている．青と橙の直線はキャッシュ有効時のテーブルの測定結果に対する回帰直線（$y = 0.031x^{0.784}$）とビューの測定結果に対する回帰直線（$y = 0.020x^{0.900}$）で表す．

はデータ総数と実行時間との間に正の相関がみられる．重複行が削除される前の検索ヒット数と実行時間との関係でも同様に正の相関がみられ，データ点数が少ないがキャッシュ有効時のビューに対する測定結果は比例関係に近い（図7）．カレンダー検索においても，検索ヒット数の増加が実行時間の増加に影響を与えていると考えられる．なお，10^8フレーム以上のテーブルで検索ヒット数が増加した理由は，データを観測日ではなくframe_id昇順にテーブルへ入力したためである．frame_idは，接頭辞TMQにカメラモジュール番号（1–4），日付，通し番号の数字が続く[8]．そのため何らかの理由でモジュール4のみでデータが取得された場合，そのデータはframe_id昇順では後方に位置することとなる．テーブルのデータ総数を増加させていくと，ある時点でそのフレームがテーブルに含まれ検索の対象となる．

検索結果数を100件に制限した場合の実行時間，及び100件に制限しない場合の実行時間に対する制限した場合の実行時間の比を表9，表10，表11に示す．ピンポイント検索とラフ検索では実行時間の比がほとんどの場合で1に近く，検索結果数の制限は実行時間の短縮に寄与しなかった．カレンダー検索では実行時間の比が1から外れる場合もあったが，その場合も比が小さくなる場合と大きくなる場合とがあり，実行時間の短縮に効果があるとは言えなかった．

上記のピンポイント検索，ラフ検索，カレンダー検索それぞれについてのデータ総数と実行時間，及び検索ヒット数と実行時間との関係についての考察，即

表9　検索結果の数に制限をかけたときのピンポイント検索のデータ総数と実行時間．実行時間の単位はmsであり，rは検索結果数を制限しない場合の実行時間に対する制限した場合の実行時間の比を表す．

	テーブル				ビュー			
データ総数	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	
	実行時間	r	実行時間	r	実行時間	r	実行時間	r
50,000	64.3	0.96	0.8	1.02	150.2	1.27	1.1	1.02
100,000	133.1	1.16	1.5	1.02	230.1	1.41	2.1	0.99
200,000	306.9	1.08	2.5	1.03	491.4	1.07	3.7	1.00
400,000	346.6	1.02	4.0	1.02	745.7	1.25	5.5	0.98
800,000	554.2	1.02	6.6	1.02	1412.2	1.27	9.2	0.99
1,600,000	1082.8	1.04	9.0	1.01	2224.8	1.02	12.1	0.99
3,200,000	1246.1	1.00	9.4	0.98	2362.7	1.04	13.7	0.99
6,400,000	2540.3	1.01	14.1	0.97	5615.9	1.00	19.9	1.00
12,800,000	6668.5	1.01	39.1	0.97	14676.9	1.00	53.3	0.98
27,918,958	17839.8	1.02	70.2	0.97	32644.7	0.99	93.4	0.97

表10　検索結果の数に制限をかけたときのラフ検索のデータ総数と実行時間．実行時間の単位はmsであり，rは検索結果数を制限しない場合の実行時間に対する制限した場合の実行時間の比を表す．

	テーブル				ビュー			
データ総数	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	
	実行時間	r	実行時間	r	実行時間	r	実行時間	r
50,000	34.5	1.20	0.5	0.99	13.9	1.00	0.5	0.98
100,000	37.0	1.07	0.5	1.00	18.3	1.00	0.5	1.02
200,000	201.1	1.12	12.1	0.99	172.3	0.74	12.5	1.00
400,000	581.4	1.09	94.5	1.00	665.1	1.00	95.7	1.01
800,000	1338.5	1.09	178.9	0.99	1717.2	1.05	185.0	1.00
1,600,000	3125.2	0.96	478.6	1.00	4046.7	1.01	523.7	1.06
3,200,000	5452.9	1.04	693.5	1.00	7006.3	1.00	726.9	1.00
6,400,000	20042.8	1.00	1129.1	0.99	23999.0	0.99	1181.4	1.00
12,800,000	53833.0	1.01	1996.5	1.01	61590.9	1.01	2098.8	1.01
27,918,958	39484.9	1.01	5091.1	1.01	81421.6	1.02	5291.4	0.99

表11　検索結果の数に制限をかけたときのカレンダー検索のデータ総数と実行時間．実行時間の単位はmsであり，rは検索結果数を制限しない場合の実行時間に対する制限した場合の実行時間の比を表す．

	テーブル				ビュー			
データ総数	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	
	実行時間	r	実行時間	r	実行時間	r	実行時間	r
50,000	14.6	0.50	0.1	0.98	14.4	1.00	0.1	1.13
100,000	31.1	2.46	0.1	1.14	16.7	1.31	0.1	1.03
200,000	22.8	1.06	0.1	1.10	13.4	0.61	0.1	0.76
400,000	32.7	0.89	0.1	1.29	18.7	1.00	0.1	0.77
800,000	27.0	1.51	0.1	1.23	17.3	1.00	0.1	1.01
1,600,000	131.5	0.82	53.4	0.86	1378.3	1.01	1142.6	1.01
3,200,000	135.0	0.88	51.4	0.82	219.5	1.00	129.2	0.98
6,400,000	184.8	1.17	50.5	0.81	225.0	1.01	127.2	0.96
12,800,000	309.9	0.93	121.8	0.98	491.4	0.97	241.1	0.98
27,918,958	470.3	0.78	150.9	0.81	852.9	0.90	435.9	0.97

ち，ラフ検索とカレンダー検索のデータ総数と実行時間との関係において検索ヒット数が0件の時の実行時間がほとんど変わっていないこと，カレンダー検索において検索ヒット数の増加時に実行時間が増加していること，キャッシュ有効時の検索ヒット数と実行時間との間に比例関係に近い関係がみられることから，SMOKA/Tomo-e Gozenシステムのテーブルとビューに対するSQLクエリの実行時間は，テーブルのデータ数ではなく検索条件に一致するデータの数に影響されると考えられる．ピンポイント検索とラフ検索では観測期間が長くなりデータ総数が増えると，同じ天域のデータ数が増えるため実行時間が長くなると推定される．カレンダー検索では一晩で取得されるデータ数には限りがあるため，データ総数が増えても実行時間にはほとんど影響がないと推定される．

キャッシュが有効時の実行時間が無効時と比較して短縮されたことは予想された通りの結果であったが，当実験環境では最大で二桁実行時間が短縮されることが分かった．

4　キャッシュ無効時の検索の高速化

約2800万フレームのスタック済みデータを入力したテーブルのテーブルファイルとインデックスファイルの総量は約53 GB（表5）である．2022年10月時点のSMOKA/Tomo-e Gozenシステムの計算機のメモリ容量は64 GBであり，全てのテーブルファイルとインデックスファイルをメモリにキャッシュすることができる．しかしスタック済みデータのフレーム数がこれまで（図8）と同じ割合で増加していくと，観測日が2021年10月頃のスタック済みデータを公開する2024年10月頃には，全てのデータをメモリにキャッシュできなくなる．そのため3章の実験でキャッシュ有効時に検索を高速におこなえることが示されたが，キャッシュ無効時の高速化方法についても調査しなければならない．本節では検索の高速化に有効と思われる，テーブル分割[21]，パラレルクエリ[22]，インデックスファイルのSSDへの配置の実験をキャッシュが無効の状態でおこない，それぞれの効果を調査した．またキャッシュ無効時との比較実験として，そして現在のSMOKA/Tomo-e Gozenシステムでの最適な設定を知るために，キャッシュ有効時でも同じ実験をおこなった．

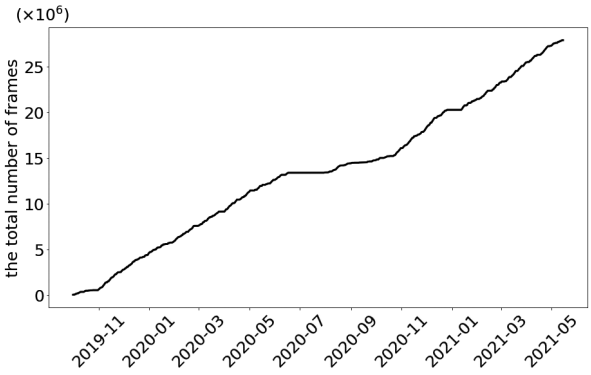

図8　本実験で使用したスタック済みデータのフレーム数の観測日に対する累積グラフ．一日平均4.7万フレームずつ増加している．

本章の実験でも3章の実験用システムを使用した．実験には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（SQL 4，SQL 5，SQL 6）を使用した．キャッシュ無

効時と有効時の実行時間をそれぞれ3回測定し，その平均と標準偏差を表14から表20に示した．SQLクエリの問い合わせ先として使用したテーブルとビューはtbl_fheaderとview_tomoe，及びそれらを赤経並びに観測日で分割したテーブルとそれら分割テーブルを参照するビューである．テーブル分割の詳細は4.1節で述べる．

4.1 テーブル分割

SMOKA/Tomo-e Gozenシステムにおけるテーブル分割の効果を検証するため，tbl_fheaderに対して数種類のテーブル分割をおこない，分割テーブル及び分割テーブルを含むビューに対するピンポイント検索，ラフ検索，カレンダー検索の実行時間が分割方法によってどのように変化するのか調査した．

テーブル分割をおこなうと子テーブルの数だけテーブルファイルが作成され，各テーブルファイルには任意の分割条件に従ったデータが集約される．Index Scanがおこなわれると，問い合わせの検索条件とテーブルの分割条件が一致する場合，問い合わせに必要な子テーブルのテーブルファイルのみが検索対象となる．子テーブルのテーブルファイルには同じ分割条件のデータが集約されているため，1ページ[23]に含まれる検索条件に一致する行の数が非分割テーブルよりも増加し，読み込むページ数が非分割テーブルよりも減少することで実行時間が短縮されるであろうと期待される．またパラレルクエリ使用時には複数の子テーブルに対し同時に検索が実行されることで実行時間が短縮されるであろうと期待される．

分割テーブルの作成

本実験では範囲分割とハッシュ分割[21]を使ってtbl_fheaderを分割した．前者は任意の列（分割キーと呼ばれる）が持つ値を複数の範囲に分割することで，データを各子テーブルに分配する方法である．親テーブルの作成にはPARTITION BY RANGE句を，子テーブルの作成にはPARTITION OF parent_table FOR VALUES FROM ('a') TO ('b')句を使用した．ここでaとbは子テーブルに分配されるデータの範囲を指定するものである．後者は分割キーの値のハッシュ値を分割数で除した際の剰余から，データの分配先となる子テーブルを決定する方法である．データは各子テーブルにランダムに分配され，各子テーブルが持つ行数は均一になる．親テーブルの作成にはPARTITION BY HASH句を，子テーブルの作成にはPARTITION OF parernt_table FOR VALUES WITH (MODULUS x, REMAINDER y)句を使用した．ここでxは分割数，yは分配先を決定するための剰余を表す．

表12は作成した分割テーブルの一覧である．範囲分割をおこなった分割テーブルを四つ，ハッシュ分割をおこなった分割テーブルを二つ作成した．

範囲分割をおこなった分割テーブルでは，crval1列（赤経）またはdate_obs列（観測日）を分割キーとしてテーブル分割をおこなった．crval1列を分割キーとしたもの（親テーブル名にraが含まれる）では，赤経を10度毎に区切り（$0° \leq$ crval1 $< 10°$, $10° \leq$ crval1 $< 20°$, ... , $350° \leq$ crval1 $< 360°$），それぞれの区間のデータが格納される36個の子テーブルを作成した．date_obs列を分割キーとしたもの（親テーブル名にdateが含まれる）では，観測日が2019-10-01から2021-05-14までのデータを一ヶ月毎に区切り（2019-10 ≤ date_obs < 2019-11, 2019-11 ≤ date_obs < 2019-12, ... , 2021-05 ≤ date_obs < 2021-06），それぞれの区間のデータが格納される20個の子テーブルを作成した．子テーブルに含まれる平均行数はcrval1列を分割キーとしたものが~775,527 ± 22,735行，date_obs列を分割キーとしたものが~1,395,948 ± 645,912行である．crval2列（赤緯）を分割キーとする方法も考えられるが，$0° \leq$ dec $< 40°$の区間に全データの75％にあたる21,180,556フレームのデータが存在し（図1左参照），データを子テーブル間で均等に分けることが難しいため採用しなかった．crval1列を分割キーとした分割テーブルの親テーブルにはframe_id列とcrval1列の複合主キー制約を，date_

表12　作成した分割テーブルの一覧.

親テーブル	分割方法	分割キー	分割範囲	分割数	子テーブル配置	被参照ビュー
tbl_fheader_ra_1d	範囲	crval1	赤経10度毎	36	単一のディスク	view_tomoe_ra_1d
tbl_fheader_ra_6d	範囲	crval1	赤経10度毎	36	六つのディスク	view_tomoe_ra_6d
tbl_fheader_date_1d	範囲	date_obs	1ヶ月毎	20	単一のディスク	view_tomoe_date_1d
tbl_fheader_date_6d	範囲	date_obs	1ヶ月毎	20	六つのディスク	view_tomoe_date_6d
tbl_fheader_hash_1d	ハッシュ	frame_id	N/A	36	単一のディスク	view_tomoe_hash_1d
tbl_fheader_hash_6d	ハッシュ	frame_id	N/A	36	六つのディスク	view_tomoe_hash_6d

obs列を分割キーとしたものにはframe_id列とdate_obs列の複合主キー制約を課した．複合主キーを採用した理由は，主キー制約を含む一意性制約を分割テーブルに課す場合は，制約に分割キーを含めなければならないためである[21]．インデックスの設定はtbl_fheaderと同様である（SQL 1参照）．

ハッシュ分割をおこなった分割テーブルでは，frame_id列を分割キーとしてテーブル分割をおこなった．範囲分割を使ったものと性能を比較できるよう，crval1列を分割キーとした場合と同様に36個の子テーブルを用意した．子テーブルに含まれる平均行数は，tbl_fheader_hash_1dが~775,522 ± 801行，tbl_fheader_hash_6dが~775,529 ± 795行である．主キー制約をframe_id列に課した．インデックスの設定はtbl_fheaderと同様である（SQL 1参照）．

crval1列，date_obs列，frame_id列を分割キーとした分割テーブルそれぞれで，子テーブルを単一のディスク上に配置したもの（親テーブル名に1dが含まれる）と六つのディスクに分散配置したもの（親テーブル名に6dが含まれる）を作成した．前者はディスクhdd1に全てのテーブルファイルとインデックスファイルを配置した．後者は六つのディスク（hdd2，hdd3，hdd4，hdd5，hdd6，hdd7）に子テーブルのテーブルファイルを分散配置し，親テーブルのテーブルファイルとインデックファイルをhdd1に配置した．各ディスク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テーブル空間を指定することで，任意のディスクにテーブルファイルとインデックスファイルを配置した．

表13は六つのディスクへの子テーブルの配置方法である．tbl_fheader_ra_6dとtbl_fheader_date_6dでは分割範囲を，tbl_fheader_hash_6dではデータの分配先を決定するために使用した剰余を，実際の子テーブルの名前の代わりとして示している．tbl_fheader_ra_6dとtbl_fheader_date_6dでは隣り合った区間の子テーブルが異なるディスク上にあるため，検索範囲が複数の子テーブルを跨ぐ場合にディスクアクセスを分散させられるであろうと期待される．tbl_fheader_hash_6dでは各子テーブルにデータがランダムに格納されているため，子テーブルのディスクへの配置方法は検索性能に影響しないと予想される．

作成した六つの分割テーブルとtbl_filemngを参照する，view_tomoeと同じ構造を持つ六つのビューを作成した（表12の被参照ビュー）．

本実験のために作成した分割テーブル及び分割テーブルを含むビューのそれぞれに対してCone Searchをおこなうユーザ定義関数（SQL 7参照）を用意し，ラフ検索のSQLクエリ（SQL 5）の問い合わせ先とした．

分割テーブルに対するクエリ実行結果

表14は分割テーブルに対するピンポイント検索，ラフ検索，カレンダー検索の実行時間の一覧である．比較のため非分割テーブルであるtbl_fheaderの結果も掲載した．なおmax_parallel_workers_per_gather = 0 [24]としているため，SQLクエリ実行時にプロセスは直列で実行されている．

適切な分割方法と分割キーを選択した場合において，

表13　六つのディスクへの子テーブルの配置方法.

親テーブル	hdd2	hdd3	hdd4	hdd5	hdd6	hdd7
tbl_fheader_ra_6d	0–10	10–20	20–30	30–40	40–50	50–60
	60–70	70–80	80–90	90–100	100–110	110–120
	120–130	130–140	140–150	150–160	160–170	170–180
	180–190	190–200	200–210	210–220	220–230	230–240
	240–250	250–260	260–270	270–280	280–290	290–300
	300–310	310–320	320–330	330–340	340–350	350–360
tbl_fheader_date_6d	2019-10	2019-11	2019-12	2020-01	2020-02	2020-03
	2020-04	2020-05	2020-06	2020-07	2020-08	2020-09
	2020-10	2020-11	2020-12	2021-01	2021-02	2021-03
	2021-04	2021-05				
tbl_fheader_hash_6d	0	1	2	3	4	5
	6	7	8	9	10	11
	12	13	14	15	16	17
	18	19	20	21	22	23
	24	25	26	27	28	29
	30	31	32	33	34	35

表14　分割テーブルに対するラフ検索，ピンポイント検索，カレンダー検索の実行時間の測定結果．実行時間の単位はmsであり，括弧内は値は標準偏差である．rは非分割テーブルであるtbl_fheaderの実行時間に対する比を表す．

検索方法	テーブル	キャッシュ無効		r	キャッシュ有効		r
ピンポイント検索	tbl_fheader	17,770.8	(23.9)	1.0	72.8	(0.4)	1.0
	tbl_fheader_ra_1d	5,576.9	(19.0)	0.31	71.9	(0.6)	0.99
	tbl_fheader_ra_6d	5,374.9	(12.2)	0.30	72.4	(1.0)	1.0
	tbl_fheader_date_1d	7,687.2	(40.9)	0.43	70.9	(0.4)	0.97
	tbl_fheader_date_6d	10,899.1	(199.9)	0.61	71.4	(0.1)	0.98
	tbl_fheader_hash_1d	22,698.5	(48.7)	1.3	75.2	(2.0)	1.0
	tbl_fheader_hash_6d	30,743.4	(1304.5)	1.7	71.9	(0.2)	0.99
ラフ検索	tbl_fheader	39,382.0	(268.2)	1.0	5,201.1	(4.0)	1.0
	tbl_fheader_ra_1d	16,082.1	(235.4)	0.41	5,231.4	(8.2)	1.0
	tbl_fheader_ra_6d	16,619.8	(389.7)	0.42	5,226.7	(3.5)	1.0
	tbl_fheader_date_1d	39,451.1	(121.7)	1.0	5,283.5	(4.7)	1.0
	tbl_fheader_date_6d	42,613.3	(333.7)	1.1	5,254.4	(2.8)	1.0
	tbl_fheader_hash_1d	128,148.7	(132.9)	3.3	5,373.5	(12.5)	1.0
	tbl_fheader_hash_6d	130,996.7	(287.1)	3.3	5,271.9	(7.7)	1.0
カレンダー検索	tbl_fheader	580.5	(19.9)	1.0	198.9	(1.5)	1.0
	tbl_fheader_ra_1d	1,355.5	(30.8)	2.3	163.6	(0.3)	0.82
	tbl_fheader_ra_6d	2,229.9	(266.9)	3.8	163.7	(0.2)	0.82
	tbl_fheader_date_1d	436.6	(27.6)	0.75	161.5	(0.6)	0.81
	tbl_fheader_date_6d	453.1	(8.5)	0.78	161.2	(0.8)	0.81
	tbl_fheader_hash_1d	1,528.1	(11.4)	2.6	155.2	(0.1)	0.78
	tbl_fheader_hash_6d	2,368.2	(18.6)	4.1	154.1	(1.3)	0.78

テーブル分割にはキャッシュ無効時の実行時間を非分割時よりも短くする効果があることが分かった．一方キャッシュ有効時の実行時間を短くする効果はほとんどみられなかった．

範囲分割をおこなった分割テーブルでは，キャッシュ無効時の実行時間が非分割テーブルより短縮されたものと長くなったものが現れた．ピンポイント検索とラフ検索ではcrval1列を分割キーとした分割テーブルの実行時間が，カレンダー検索ではdate_obs列を分割キーとした分割テーブルの実行時間が最も短縮された．これらのテーブルではその実行計画から，検索条件に一致するデータが格納された子テーブルのみに問い合わせがおこなわれている．ピンポイント検索とラフ検索ではcrval1列が，カレンダー検索ではdate_obs列がSQLクエリの検索条件に含まれており，SQLクエリの検索条件に分割キーが含まれたことで，一部の子テーブルのみを参照する効率的な検索がおこなわれたと考えられる．また，テーブル分割時にテーブルファイルの特定のページにデータが集約されたことでディスクから読み込まれるページ数が減少し，実行時間が短縮したものと考えられる．子テーブルを六つのディスクに分散配置した場合の実行時間は，単一のディスクに配置した場合と比べ，ピンポイント検索では~1.04倍，ラフ検索では~1.03倍，カレンダー検索では~0.96倍になった．実験結果のばらつきを考慮すると，分散配置は検索速度の変化にほとんど影響しなかった．

一方SQLクエリの検索条件に分割キーが含まれない場合，即ちピンポイント検索とラフ検索ではdate_obs列を分割キーとした分割テーブル，カレンダー検索ではcrval1列を分割キーとした分割テーブルでは，その実行計画から，全ての子テーブルをスキャンする非効率的な問い合わせがおこなわれた．その結果，ラフ検索とカレンダー検索では非分割テーブルと実行時間が同等か，あるいは長くなったものと考えられる．ピンポイント検索では実行時間が短くなったが，子テーブル間で抽出されるデータ数に偏りがあったことがその原因であると考えている．スタック済みデータは日付情報を含むファイル名昇順にテーブルへ入力したため，テーブルのデータの並びはdate_obs列昇順である．ある天域を観測する場合，天域によっては観測に適した時期は限られるため，date_obs列を分割キーとした分割テーブルではその天域が入力される子テーブルも限られる．またある日にある天域を観測するとその観測データはテーブルファイル内のページに順番

に書き込まれていくため，それらのページに特定の天域のデータが集中することになる．実際，本検索では合計3,675件のデータがdate_obs列で分けられた20個の子テーブルから抽出されるが，その抽出件数は図9のように2019年11月と2020年11月を頂点とした山形を示しており，子テーブル間で抽出されるデータ数に偏りがある．検索条件に一致するデータが存在しない子テーブルではテーブルファイルが読み込まれないため処理が高速におこなわれたこと，検索条件に一致するデータが存在する子テーブルではデータがテーブルファイル内の特定のページに集約されたことで読み込むページの数が減り，実行時間が短くなったと考えられる．

ハッシュ分割をおこなった分割テーブルでは，キャッシュ無効時の実行時間が非分割テーブルよりも長くなった．その実行計画から，全ての子テーブルをスキャンする非効率的な問い合わせがおこなわれた．子テーブルにデータがランダムに格納されたためと考えられる．

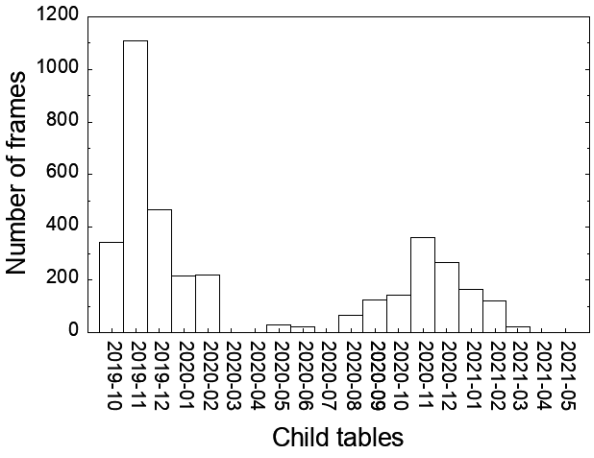

図9　date_obs列を分割キーとした分割テーブルに対するピンポイント検索（SQL 4）の子テーブルあたりのヒット数.

分割テーブルを含むビューに対するクエリ実行結果

表15は分割テーブルを含むビューに対する各検索の実行時間の一覧である．比較のため非分割テーブルtbl_fheaderを含むビューであるview_tomoeの結果も掲載した．分割テーブルに対する実行時間（表14）と比較すると実行時間がキャッシュ無効時有効時とも長くなったが，その大小関係は同じ傾向を示した．実行時間が長くなった原因として次の要因が考えられる．

表15　分割テーブルを含むビューに対するピンポイント検索，ラフ検索，カレンダー検索の実行時間の測定結果．実行時間の単位はmsであり，括弧内の値は標準偏差である．rは非分割テーブルを含むビューであるview_tomoeの実行時間に対する比を表す.

検索方法	ビュー	キャッシュ無効		r	キャッシュ有効		r
ピンポイント検索	view_tomoe	33,482.8	(252.5)	1.0	89.3	(1.1)	1.0
	view_tomoe_ra_1d	14,004.7	(277.5)	0.42	90.4	(0.6)	1.0
	view_tomoe_ra_6d	15,914.0	(443.2)	0.48	90.9	(0.5)	1.0
	view_tomoe_date_1d	17,567.6	(510.7)	0.52	89.8	(0.1)	1.0
	view_tomoe_date_6d	29,969.8	(769.8)	0.90	90.3	(0.3)	1.0
	view_tomoe_hash_1d	43,272.5	(920.7)	1.3	95.5	(1.0)	1.1
	view_tomoe_hash_6d	71,737.9	(629.1)	2.1	95.5	(1.0)	1.1
ラフ検索	view_tomoe	81,501.1	(1478.7)	1.0	5,481.0	(25.5)	1.0
	view_tomoe_ra_1d	36,533.0	(562.8)	0.45	5,466.7	(3.4)	1.0
	view_tomoe_ra_6d	38,471.7	(2,186.6)	0.47	5,516.8	(5.2)	1.0
	view_tomoe_date_1d	58,795.6	(120.8)	0.72	5,699.1	(1.2)	1.0
	view_tomoe_date_6d	65,298.3	(251.7)	0.80	5,661.8	(7.8)	1.0
	view_tomoe_hash_1d	179,785.1	(563.0)	2.2	5,785.1	(7.7)	1.1
	view_tomoe_hash_6d	189,058.1	(1,732.1)	2.3	5,705.2	(11.3)	1.0
カレンダー検索	view_tomoe	881.2	(57.9)	1.0	451.5	(0.8)	1.0
	view_tomoe_ra_1d	2,109.9	(80.5)	2.4	453.2	(1.1)	1.0
	view_tomoe_ra_6d	3,009.0	(115.4)	3.4	452.0	(1.5)	1.0
	view_tomoe_date_1d	896.7	(49.6)	1.0	456.4	(0.6)	1.0
	view_tomoe_date_6d	928.4	(43.0)	1.1	454.0	(0.9)	1.0
	view_tomoe_hash_1d	2,448.7	(24.2)	2.8	483.2	(1.7)	1.1
	view_tomoe_hash_6d	3,877.5	(661.9)	4.4	496.8	(1.2)	1.1

キャッシュ無効時の実行時間が長くなった原因は，ビューに対する検索ではtbl_fheaderに加え，tbl_filemngのテーブルファイルとインデックスファイルのデータをディスクから読み込む必要があったためと考えられる．キャッシュ有効時の実行時間が長くなった原因は，Nested Loop処理が実行計画に加わったためと考えられる．Nested LoopはテーブルAとテーブルBの各行を比較して，結合条件を満たす行同士を結合する処理である[20]．テーブルBの結合条件である列にインデックスが設定されている場合，その処理時間はテーブルAの行数とテーブルBから結合条件を満たす行を抽出するために要した時間との積になる．本実験ではtbl_fheader*がテーブルAに相当し，ピンポイント検索，ラフ検索，カレンダー検索それぞれで3,676行，80,881行，67,906行が結合対象となる．view_tomoe_ra_1dの場合，ラフ検索とカレンダー検索ではテーブルBに相当するtbl_filemngから結合条件を満たす行を抽出するために0.004 msを要している．したがってNested Loopの処理時間は300 ms程度となる．一方ピンポイント検索では抽出時間が0.010 msであるため処理時間は37 ms程度となる．ビューに対する検索ではこれらの時間が加算されたため，キャッシュ無効時有効時とも実行時間が長くなったと考えられる．

以上の結果から，テーブル分割を使ってキャッシュ無効時の実行時間を短縮するためには，範囲分割を使って分割キーがSQLクエリの検索条件に含まれるようテーブル設計すれば良いということが分かった．またキャッシュ有効時の実行時間を短縮する効果がほとんどないことも分かった．SMOKA/Tomo-e Gozenシステムにおいては，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の短縮化が期待できる一方，カレンダー検索ではテーブル分割をおこなっても実行時間は短縮されないだろう．

4.2 パラレルクエリ

ピンポイント検索，ラフ検索，カレンダー検索で使用されるSQLクエリの並列化（パラレルクエリ）による実行時間の短縮効果を調査する実験をおこなった．4.1節の実験と同様に，表14のテーブル及び表15のビューをSQLクエリの対象とした．

本実験ではPostgreSQLのパラメータ設定をforce_parallel_mode = on [24]かつmax_parallel_workers_per_gather = 8 [25]にした．パラレルクエリで実際に使用されるワーカー数は，max_parallel_workers_per_gatherの制限に基づきプランナが最適な数を決定するため，場合によってはその数が0となりパラレルクエリが実行されない[26]．パラレルクエリを強制的におこなわせるためforce_parallel_mode = onとした．また本実験で使用するユーザ定義関数のパラレルラベルを'PARALELL SAFE'に設定した[27]．ユーザ定義関数のパラレルラベルの初期値は'PARALELL UNSAFE'であり，そのままではパラレルクエリを実行できない．本実験で使用するユーザ定義関数は，テーブルのロック及び更新をおこなわないパラレル安全なものである．

表16，表17，表18はパラレルクエリを使ったピンポイント検索，ラフ検索，カレンダー検索の実行時間の一覧である．各SQLクエリの実行計画に示されるワーカー数の値並びにパラレルクエリ非使用時の実行時間に対する使用時の実行時間の比も併記した．

テーブルに対するパラレルクエリの実行結果

テーブルに対する検索では，いずれの検索でもパラレルクエリによる実行時間の短縮効果がほとんどみられなかった．キャッシュ無効時の実行時間はパラレルクエリ非使用時（表14）と比べ，ばらつきの範囲で同じであった．キャッシュ有効時の実行時間はパラレルクエリ非使用時と比べ，ピンポイント検索は40–70％程度，カレンダー検索は20–40％程度長くなった．ラフ検索ではtbl_fheader_ra*の実行時間が10％程度短くなったが，それ以外は同程度であった．ピンポイント検索とカレンダー検索では全てのテーブルでパラレルクエリのワーカー数が1であった．これらについて，プランナがワーカー数を1とした原因は不明である．パラレルクエリのワーカー数に0または1を加えた値が処理の並列度である．並列度はプランナが自動的に決定するため，ユーザがその値や制限について指定することはできない．実行計画ではワーカー数が1であった検索では並列度も1であり，並列処理がおこなわれていなかった．ラフ検索ではtbl_fheader_ra*のワーカー数が3であったが，それ以外は1であった．並列処理がおこなわれた結果，キャッシュ有効時のtbl_fheader_ra*の実行時間が短くなったものと考えられる．

ビューに対するパラレルクエリの実行結果

ビューに対する検索では，各検索でパラレルクエリによる実行時間の短縮効果が確認できた．

ピンポイント検索では，キャッシュ無効時に表16の全てのビューでパラレルクエリによる実行時間の短縮効果がみられた．キャッシュ有効時はview_tomoe_date*とview_tomoe_hash*で短縮効果がみられた．パラレルクエリ使用時の実行時間（表16）と非使用時の実行時間（表15）が最短のもの同士を比較すると，キャッシュ無効時に60％程度，キャッシュ有効時に30％程度，実行時間が短縮されていた．表16中のビュー同士を比較すると，キャッシュ無効時

はview_tomoe_date_6dが最速であり，実行時間の比が他のビューの0.3–0.8倍程度であった．キャッシュ有効時はview_tomoe_date*が最速であり，実行時間が他のビューの0.6–0.9倍程度であった．パラレルクエリ非使用時の実行時間に対する比は，ハッシュ分割を使ったview_tomoe_hash_6dがキャッシュ無効時に最も小さく，効率的にパラレルクエリが実行されたことが分かる．しかしパラレルクエリ非使用時の実行時間が大きかったため，表16のビューの中で最短にはならなかった．キャッシュ無効時に子テーブルを分散配置したビューの実行時間は単一のディスクに配置した場合の0.5–0.6倍程度となった．子テーブルの分散配置とパラレルクエリの組み合わせによる実行時間の短縮効果であると考えられる．非分割テーブルを参照するビューのワーカー数は1であったが，実行計画の並列度は2を示していた．並列処理の効果がキャッシュ無効時の実行時間の結果に現れていると考えられる．

ラフ検索では，view_tomoe_ra*でパラレルクエリによる実行時間の短縮効果がみられた．一方view_tomoe_date*ではパラレルクエリ非使用時よりも実行時間が長くなった．パラレルクエリ使用時の実行時間（表17）と非使用時の実行時間（表15）が最短のもの同士を比較すると，キャッシュ無効時に30％程度，キャッシュ有効時に20％程度，実行時間が短縮されていた．表17中のビュー同士を比較すると，キャッシュ無効時はview_tomoe_ra_6dが最速であり，実行時間が他のビューの0.1–0.8倍であった．キャッシュ有効時はview_tomoe_ra*が最速であり，実行時間の比が他のビューの0.1–0.9倍であった．ハッシュ分割を使ったビューでは並列に処理がおこなわれずパラレルクエリ非使用時と同等の実行時間となったが，その原因は不明である．キャッシュ無効時は子テーブルを分散配置したビューの実行時間が単一のディスクに配置した場合の0.8–1.1倍程度となった．ピンポイント検索の場合と同様に，子テーブルの分散配置とパラレルクエリの組み合わせによる実行時間の短縮効果がview_tomoe_ra*とview_tomoe_date*で現れたと考えられる．非分割テーブルを参照するビューのワーカー数は1であったが，実行計画の並列度は1を示しており，並列処理はおこなわれなかった．

カレンダー検索では，表18の全てのビューでパラレルクエリによる実行時間の短縮効果がみられた．パラレルクエリ使用時の実行時間（表18）と非使用時の実行時間（表15）が最短のもの同士を比較すると，キャッシュ無効時に20％程度，キャッシュ有効時に70％程度，実行時間が短縮されていた．表18中のビュー同士を比較すると，キャッシュ無効時はview_tomoe_ra_1dが2倍，view_tomoe_hash_1dが3倍，view_tomoe_hash_6dが1.3倍程度，実行時間が他のビューよりも長く，それ以外のビューの実行時間は同程度であった．キャッシュ有効時はview_tomoe_ra*とview_tomoe_hash*が最速であり，実行時間の比が他のビューの0.5–0.6倍であった．ビューのワーカー数はview_tomoe_ra*とview_tomoe_hash*が6であり，それ以外のビューは2であった．crval1列で分割すると特定の日

表16　ピンポイント検索のパラレルクエリ実行時間の測定結果．実行時間の単位はmsであり，括弧内は標準偏差．wはワーカー数．rはパラレルクエリを使用しない場合の実行時間に対する比．

	キャッシュ無効				キャッシュ有効			
テーブル	実行時間		w	r	実行時間		w	r
tbl_fheader	17,664.3	(57.4)	1	0.99	109.0	(2.2)	1	1.5
tbl_fheader_ra_1d	5,594.3	(42.5)	1	1.0	103.7	(2.6)	1	1.4
tbl_fheader_ra_6d	5,362.3	(32.1)	1	1.0	105.3	(0.9)	1	1.5
tbl_fheader_date_1d	7,708.0	(47.5)	1	1.0	110.3	(2.1)	1	1.6
tbl_fheader_date_6d	10,708.7	(34.2)	1	0.98	114.7	(0.5)	1	1.6
tbl_fheader_hash_1d	22,400.6	(160.0)	1	0.99	123.0	(1.4)	1	1.6
tbl_fheader_hash_6d	29,864.5	(1,354.5)	1	0.97	121.5	(1.9)	1	1.7
ビュー								
view_tomoe	10,961.8	(126.0)	1	0.33	96.8	(1.2)	1	1.1
view_tomoe_ra_1d	11,274.5	(199.6)	2	0.81	94.8	(1.6)	2	1.0
view_tomoe_ra_6d	6,807.6	(228.5)	2	0.43	95.8	(1.6)	2	1.1
view_tomoe_date_1d	11,510.4	(39.9)	5	0.66	60.2	(0.1)	5	0.67
view_tomoe_date_6d	5,268.8	(71.6)	5	0.18	58.4	(1.2)	5	0.65
view_tomoe_hash_1d	17,538.6	(237.9)	6	0.41	68.8	(1.1)	6	0.69
view_tomoe_hash_6d	7,950.4	(329.6)	6	0.11	70.1	(13.2)	6	0.73

表17　ラフ検索のパラレルクエリ実行時間の測定結果．実行時間の単位はmsであり，括弧内は標準偏差．wはワーカー数．rはパラレルクエリを使用しない場合の実行時間に対する比．

	キャッシュ無効				キャッシュ有効			
テーブル	実行時間		w	r	実行時間		w	r
tbl_fheader	39,383.1	(247.3)	1	1.0	5,238.5	(46.4)	1	1.0
tbl_fheader_ra_1d	18,072.2	(512.2)	3	1.1	4,883.5	(654.1)	3	0.93
tbl_fheader_ra_6d	15,951.8	(731.8)	3	0.96	4,467.3	(182.0)	3	0.85
tbl_fheader_date_1d	39,674.7	(384.5)	1	1.0	5,401.0	(9.5)	1	1.0
tbl_fheader_date_6d	42,633.5	(199.4)	1	1.0	5,363.0	(8.5)	1	1.0
tbl_fheader_hash_1d	128,238.2	(346.0)	1	1.0	5,551.8	(15.6)	1	1.0
tbl_fheader_hash_6d	130,945.9	(283.3)	1	1.0	5,492.7	(7.5)	1	1.0
ビュー								
view_tomoe	81,541.5	(1,637.6)	1	1.0	5,623.7	(9.4)	1	1.0
view_tomoe_ra_1d	30,630.1	(578.9)	3	0.84	4,612.5	(152.6)	3	0.84
view_tomoe_ra_6d	25,441.2	(539.8)	3	0.66	4,897.8	(871.7)	3	0.89
view_tomoe_date_1d	74,597.9	(3,614.8)	5	1.3	21,788.7	(524.3)	5	3.8
view_tomoe_date_6d	58,570.9	(1,301.9)	5	0.90	39,324.4	(533.4)	5	6.9
view_tomoe_hash_1d	180,418.5	(244.7)	1	1.0	5,995.2	(5.6)	1	1.0
view_tomoe_hash_6d	190,899.4	(1,036.4)	1	1.0	5,908.8	(11.5)	1	1.0

表18　カレンダー検索のパラレルクエリ実行時間の測定結果．実行時間の単位はmsであり，括弧内は標準偏差．wはワーカー数．rはパラレルクエリを使用しない場合の実行時間に対する比．

	キャッシュ無効				キャッシュ有効			
テーブル	実行時間		w	r	実行時間		w	r
tbl_fheader	588.0	(13.0)	1	1.0	239.6	(0.2)	1	1.2
tbl_fheader_ra_1d	1,398.0	(13.0)	1	1.0	214.3	(0.4)	1	1.3
tbl_fheader_ra_6d	2,794.2	(480.2)	1	1.3	212.1	(3.0)	1	1.3
tbl_fheader_date_1d	452.0	(3.4)	1	1.0	196.6	(0.6)	1	1.2
tbl_fheader_date_6d	465.6	(12.6)	1	1.0	196.2	(0.2)	1	1.2
tbl_fheader_hash_1d	1,572.1	(12.9)	1	1.0	221.3	(3.1)	1	1.4
tbl_fheader_hash_6d	2,368.4	(74.6)	1	1.0	216.5	(3.0)	1	1.4
ビュー								
view_tomoe	723.8	(12.6)	2	0.82	299.1	(0.8)	2	0.66
view_tomoe_ra_1d	1,540.3	(38.0)	6	0.73	154.6	(2.2)	6	0.34
view_tomoe_ra_6d	788.0	(20.4)	6	0.26	152.3	(1.0)	6	0.34
view_tomoe_date_1d	692.9	(73.6)	2	0.70	251.3	(7.3)	2	0.55
view_tomoe_date_6d	712.7	(91.1)	2	0.77	254.9	(2.1)	2	0.56
view_tomoe_hash_1d	2,267.5	(32.0)	6	0.93	157.3	(0.5)	6	0.33
view_tomoe_hash_6d	960.0	(8.6)	6	0.25	163.9	(0.7)	6	0.33

付のデータがより多くの子テーブルに分散されるため，date_obs列を検索条件とするカレンダー検索の並列度が大きくなったと考えられる．ピンポイント検索とラフ検索でも同様の理由で分割キーが検索条件に含まれない時に並列度が大きくなったと考えられる．ハッシュ分割ではデータがランダムに子テーブルに分散されているため並列度が大きくなったと考えられる．

以上の結果から，ビューを検索対象としたSMOKA/Tomo-e Gozenシステムではパラレルクエリを用いることで実行時間を短縮できることが示された．今後データをメモリにキャッシュできなくなった際，ピンポイント検索ではview_tomoe_date_6dが，ラフ検索ではview_tomoe_ra_6dが，カレンダー検索ではview_tomoe_ra_1d及びview_tomoe_hash*を除いたものがパ

ラレルクエリに最適なビューとなるだろう．キャッシュが有効な現在のSMOKA/Tomo-e Gozenシステムでもパラレルクエリを用いることで実行時間を短縮でき，ピンポイント検索ではview_tomoe_date*が，ラフ検索とカレンダー検索ではview_tomoe_ra*が最適なビューとなる．

4.3 インデックスファイルのSSDへの配置

3章の実験では検索一致数が0件，すなわちテーブルファイルの読み込みがおこなわれない場合においてもキャッシュ無効時と有効時の実行時間に差が生じた（表7，表8）．ラフ検索，ピンポイント検索，カレンダー検索ではIndex Scanがおこなわれているため，実行時間の差はインデックスファイルを読み込むために要した時間であると考えられる．

インデックスファイルの読み込み時間を短縮するために，HDDよりも高速に読み書き可能なSSDを実験用計算機に追加し，SSD上にインデックスファイルを配置して，テーブル及びビューに対するラフ検索，ピンポイント検索，カレンダー検索の実行時間を測定した．SSD上へのインデックスファイルの配置が，各検索のテーブル及びビューに対する実行時間の短縮にどの程度の効果があるのか調査した．

インデックスファイルを配置するために用意したSSDは容量が200 GBのSATA接続（6 Gbps）のものである．実験用計算機に接続可能なディスクの本数は最大8本であるが，既に全てのSATAポートを使用していたため接続中のHDD（hdd7）を取り外してSSDを接続した．このため本実験では子テーブルを六つのディスクに分散配置した場合の分割テーブルは扱わなかった．

作成したテーブルはtbl_fheader_ssdidx，tbl_fheader_ra_1d_ssdidx，tbl_fheader_date_1d_ssdidxである．それぞれのテーブルはtbl_fheader，tbl_fheader_ra_1d，tbl_fheader_date_1dのテーブルファイルをディスクhdd1上に，インデックスファイルをSSD上に配置したものである．SSD上にテーブル空間を作成し，インデックス作成時にテーブル空間を指定することで，SSD上にインデックスファイルを配置した．これらのテーブルとtbl_filemngのインデックスファイルをSSD上に配置したtbl_filemng_ssdidxを使って，view_tomoe_ssdidx，view_tomoe_ra_1d_ssdidx，view_tomoe_date_1d_ssdidxを作成した．なお4.1節と4.2節の実験から，ハッシュ分割はSMOKA/Tomo-e Gozenシステムに不適切ということが分かったため本実験の対象からは除外した．

表19と表20はSSD上にインデックスファイルを配置したテーブルとSSD上にインデックスファイルを配置したテーブルを含むビューそれぞれに対するピンポイント検索，ラフ検索，カレンダー検索の実行時間の一覧である．HDD上にインデックスファイルを配置したとき（表14，表15）と比べると，キャッシュ有効時は実行時間を短くする効果がほとんどみられなかった．一方キャッシュ無効時は実行時間が短くなったものと長くなったものがみられた．tbl_fheader_ssdidxに対するラフ検索とカレンダー検索では実行計画のスキャン方式がIndex ScanからBitmap Index Scanに変わったが，それ以外の検索ではHDD上にインデックスファイルを配置した場合と同じ実行計画であった．実行時間の変化と実行計画の間には関係がないと思われる．

キャッシュ無効時における各検索の実行時間が最短であったテーブルについて，インデックスファイルをHDD上に配置した場合とSSD上に配置した場合とを比べると，ピンポイント検索ではtbl_fheader_ra_6dに対するtbl_fheader_date_1d_ssdidxの実行時間が85％程度，ラフ検索ではtbl_fheader_ra_1dに対するtbl_fheader_ra_1d_ssdidxの実行時間が15％程度，カレンダー検索ではtbl_fheader_date_1dに対するtbl_fheader_date_1d_ssdidxの実行時間が25％程度短縮された．インデックスファイルをSSD上に配置することで分割テーブルに対するSQLクエリの実行時間を短縮できることが分かった．

同様に実行時間が最短のビュー同士を比較すると，ピンポイント検索ではview_tomoe_ra_1dに対するview_tomoe_date_1d_ssdidxの実行時間が30％程度短縮されたが，ラフ検索及びカレンダー検索の最短の実行時間は同程度であった．キャッシュ無効時のビューに対してはパラレルクエリが実行時間の短縮に効果があることが4.2節で分かっており，ピンポイント検索については，インデックスファイルをSSD上に配置するよりもパラレルクエリを有効にするほうが最短実行時間が50％短い．ビューに対してはインデックスファイルをSSD上に配置するよりもパラレルクエリを有効にするほうが適切であると言える．

表21はキャッシュ無効時の，HDD上にインデックスファイルを配置した時の実行時間（表14，表15）に対するSSD上にインデックスファイルを配置した時の実行時間（表19，表20）の比である．テーブルを対象とした検索の場合，ピンポイント検索とラフ検索ではtbl_fheader_date_1d_ssdidxが，カレンダー検索ではtbl_fheader_ra_1d_ssdidxの比の値が最も小さくなった．これらは4.1節で述べた分割キーがSQLクエリの検索条件に含まれない組み合わせである．全ての子テーブルに対して問い合わせが実行されるため，それぞれの子テーブル毎にインデックスファイルの読み込みが発生する．インデックスファイルの読み込み回数が多い分，SSD上に配置した効果が大きく出たものと推測し

表19　SSD上にインデックスを配置したテーブルに対するピンポイント検索，ラフ検索，カレンダー検索の実行時間（単位はms，括弧内の値は標準偏差）.

検索方法	テーブル	キャッシュ無効		キャッシュ有効	
ピンポイント検索	tbl_fheader_ssdidx	20867.7	(181.5)	73.1	(0.9)
	tbl_fheader_ra_1d_ssdidx	4974.2	(20.4)	75.0	(0.7)
	tbl_fheader_date_1d_ssdidx	720.4	(59.9)	73.3	(0.0)
ラフ検索	tbl_fheader_ssdidx	23393.0	(363.9)	5207.7	(14.5)
	tbl_fheader_ra_1d_ssdidx	13054.1	(480.7)	5059.7	(4.0)
	tbl_fheader_date_1d_ssdidx	13247.5	(863.6)	5277.2	(12.7)
カレンダー検索	tbl_fheader_ssdidx	720.7	(80.0)	194.1	(3.6)
	tbl_fheader_ra_1d_ssdidx	946.9	(59.2)	153.0	(3.6)
	tbl_fheader_date_1d_ssdidx	328.1	(52.6)	150.8	(0.4)

表20　SSD上にインデックスを配置したテーブルを含むビューに対するピンポイント検索，ラフ検索，カレンダー検索の実行時間（単位はms，括弧内の値は標準偏差）.

検索方法	ビュー	キャッシュ無効		キャッシュ有効	
ピンポイント検索	view_tomoe_ssdidx	40419.7	(1122.2)	98.6	(1.9)
	view_tomoe_ra_1d_ssdidx	16987.4	(318.3)	99.4	(1.1)
	view_tomoe_date_1d_ssdidx	10000.5	(1137.7)	95.7	(0.1)
ラフ検索	view_tomoe_ssdidx	47088.4	(1860.6)	5573.1	(5.5)
	view_tomoe_ra_1d_ssdidx	38737.2	(576.5)	5461.3	(12.2)
	view_tomoe_date_1d_ssdidx	42823.4	(971.5)	5558.3	(2.7)
カレンダー検索	view_tomoe_ssdidx	1178.7	(65.5)	469.3	(1.3)
	view_tomoe_ra_1d_ssdidx	1727.8	(58.1)	466.6	(2.2)
	view_tomoe_date_1d_ssdidx	873.4	(122.8)	467.2	(4.2)

表21　キャッシュ無効時におけるHDDにインデックスを配置した時の実行時間に対するSSDにインデックスを配置した時の実行時間の比.

	ピンポイント検索	ラフ検索	カレンダー検索
tbl_fheader_ssdidx	1.17	0.59	1.24
tbl_fheader_ra_1d_ssdidx	0.89	0.81	0.70
tbl_fheader_date_1d_ssdidx	0.09	0.34	0.75
view_tomoe_ssdidx	1.21	0.58	1.34
view_tomoe_ra_1d_ssdidx	1.21	1.06	0.82
view_tomoe_date_1d_ssdidx	0.57	0.73	0.97

ている．分割キーがSQLクエリの検索条件に含まれる組み合わせである，tbl_fheader_ra_1d_ssdidxに対するピンポイント検索とラフ検索，tbl_fheader_date_1d_ssdidxに対するカレンダー検索でもHDD上に配置した時と比べて実行時間が短縮されており，分割テーブルに対する検索では高速に読み書き可能な領域にインデックスファイルを配置することで検索速度を高速化できると考えられる．ビューを対象とした検索の場合，テーブルを対象とした場合と同様に分割キーとなる列がSQLクエリの検索条件に含まれない組み合わせで実行時間が短縮された．一方，分割キーとなる列がSQLクエリの検索条件に含まれる組み合わせではピンポイント検索とラフ検索の実行時間が長くなった．非分割のテーブルとビューにおいてもピンポイント検索とカレンダー検索の実行時間が長くなったが，本論文執筆時点で原因は不明である．

以上の結果から，分割キーとした列がSQLクエリの検索条件に含まれないようテーブルを分割し，そ

のインデックスファイルをSSD上へ配置することで，テーブルに対する各SQLクエリの実行時間を最も短縮できることが分かった．同方法でビューに対する各SQLクエリの実行時間も短縮できることが分かった．本実験では当該領域としてSSDを利用したが，インデックスは二次的に生成可能なデータであることから高速なRAMディスクへインデックスファイルを配置するという方法も考えられるだろう．

5 まとめ

本論文では巨大化した観測データがデータベースの検索速度に与える影響を知るために，SMOKA/Tomo-e Gozenシステムのデータベースと約2800万件のスタック済みデータ及び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クエリを利用して，公開されるデータ量が増えた際にテーブルとビューに対する検索用クエリの実行時間がどのように変化するのか調査した．結論は以下の通りである．

・SQLクエリの実行時間は，テーブルのデータ数ではなく検索条件に一致するデータの数に影響されると考えられる．そのため観測期間が長くなりデータ総数が増加すると，ピンポイント検索とラフ検索の実行時間が増える一方，カレンダー検索の実行時間はほとんど増えないものと推定される．
・キャッシュ有効時のSQLクエリの実行時間が無効時と比べて最大で二桁短縮される．

次に同システムにおいてキャッシュ無効時におけるSQLクエリの実行時間の短縮に有効と思われる，テーブル分割，パラレルクエリ，インデックスファイルのSSD上への配置の実験をおこない，それぞれの有効性について調査した．

SQLクエリがテーブルを問い合わせ先とする場合の結論は以下の通りである．

・分割キーとした列がSQLクエリの検索条件に含まれるよう範囲分割を使ったテーブル分割をおこなうことで，SQLクエリの実行時間を短縮できる．
・パラレルクエリを有効化してもSQLクエリの実行時間は短縮できない．
・分割キーとした列がSQLクエリの検索条件に含まれないよう範囲分割を使ったテーブル分割をおこない，そのインデックスファイルをSSD上へ配置することで各SQLクエリの実行時間を最も短縮できる．

SQLクエリがビューを問い合わせ先とする場合の結論は以下の通りである．

・ピンポイント検索とラフ検索では，分割キーとした列がSQLクエリの検索条件に含まれるよう範囲分割を使ったテーブル分割をおこなうことで，SQLクエリの実行時間を短縮できる．
・パラレルクエリを有効化することでSQLクエリの実行時間を短縮できる．ピンポイント検索ではview_tomoe_date_6dが，ラフ検索ではview_tomoe_ra_6dが問い合わせ先として最適であり，カレンダー検索ではview_tomoe_ra_1d及びview_tomoe_hash*を除いたものが問い合わせ先として最適である．
・分割キーとした列がSQLクエリの検索条件に含まれないよう範囲分割を使ったテーブル分割をおこない，そのインデックスファイルをSSD上へ配置することで各SQLクエリの実行時間を短縮できる．

また，ハッシュ分割を使ったテーブル分割は，パラレルクエリ非使用時の実行時間が長くなることが分かった．パラレルクエリ使用時は並列に検索がおこなわれることで実行時間が改善されるものの，範囲分割を使ったテーブル分割の実行時間には及ばなかった．

SMOKA/Tomo-e Gozenシステムでは，2024年10月頃には公開データの全てのテーブルファイルとインデックスファイルをメモリにキャッシュできなくなる．その場合には，上記のビューの結論に基づき，tmq_kisoテーブルの分割，分割した子テーブルのテーブルファイルの複数ディスクへの分散配置，及びパラレルクエリの有効化を施すことで検索速度を高速化できるだろう．キャッシュが有効である現時点のSMOKA/Tomo-e Gozenシステムにおいても，各検索での最適な問い合わせ先についてはキャッシュ無効時と異なるが，同様の設定により検索速度を高速化できるだろう．同システムの計算機はレンタル機器であるため既存のHDDをSSDに交換してインデックスファイルを配置することは難しいが，USB接続のSSDやRAMディスクを利用した検索速度の高速化が検討できるだろう．

本論文では，巨大化した観測データがデータベースの検索速度に与える影響，そしてキャッシュ無効時の検索速度を向上させるための一定の知見を得ることができたが，次に述べるものが課題として残る．本論文でおこなった各実験では，ラフ検索とピンポイント検索の検索条件として赤経・赤緯のみを指定し，その実行速度を測定した．実際の環境では観測日と赤経・赤緯の両方を検索条件に含む検索もおこなわれるため，その場合の実験をおこない最適な設定を見つけなければならない．テーブルの分割数も一通りの値のみで実

験をおこなったため，値を変更して最適な設定を見つけなければならない．SSD使用時のパラレルクエリの実験や，テーブルファイルをSSD上に配置した場合の実行時間の検証，そしてSSDへのインデックスファイルの配置実験で実行速度が低下したテーブルとビューが現れた原因についても調査が必要だろう．座標検索に使用する検索インデックスとして複合インデックスを利用しているが，PostGIS [28]等による球面インデックスの利用についても調査が必要だろう．

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．本論文の執筆にあたって多大なる助言を天文データセンター高田唯史氏にいただいた．両氏にはここで厚く感謝する．匿名の査読者には多くの貴重なご指摘ご意見をいただいた．ここで厚く感謝する．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．

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, 西村高徳, 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] Sako, S. et al.: The Tomo-e Gozen wide field CMOS camera for the Kiso Schmidt telescope, *Proc. SPIE*, **10702**, 107020J (2018).

[10] Matsubayashi, K: Triccs ホームページ, 2022, http://www.o.kwasan.kyoto-u.ac.jp/inst/triccs/, (最終閲覧日: 2022-12-06).

[11] PostgreSQL グローバル開発グループ:「11.2. インデックスの種類」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/indexes-types.html, (最終閲覧日 : 2022-08-29).

[12] PostgreSQL グローバル開発グループ:「11.3. 複数列インデックス」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/indexes-multicolumn.html, (最終閲覧日 : 2022-08-29).

[13] PostgreSQL グローバル開発グループ:「11.7. 式インデックス」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/indexes-expressional.html, (最終閲覧日 : 2022-08-29).

[14] PostgreSQL グローバル開発グループ:「11.1. 序文」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/indexes-intro.html, (最終閲覧日 : 2022-08-30).

[15] Power, R. A.: Large Catalogue Query Performance in Relational Databases, *Publications of the Astronomical Society of Australia*, **24**, 13–20 (2007).

[16] PostgreSQL グローバル開発グループ:「40.3. マテリアライズドビュー」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/rules-materializedviews.html, (最終閲覧日 : 2022-08-30).

[17] 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: 2022-08-30).

[18] PostgreSQL グローバル開発グループ:「50.5. プランナ/オプティマイザ」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/planner-optimizer.html, (最終閲覧日: 2022-08-30).

[19] PostgreSQL グローバル開発グループ:「11.5. 複数のインデックスの組み合わせ」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/indexes-bitmap-scans.html, (最終閲覧日: 2022-08-31).

[20] 鈴木啓修: PostgreSQL 全機能バイブル: 現場で役立つ A to Z, 技術評論社, (2012).

[21] 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: 2022-08-31).

[22] 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: 2022-08-31).

[23] 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: 2022-08-31).

[24] PostgreSQL グローバル開発グループ:「19.7. 問い合わせ計画」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/12/html/runtime-config-query.html, (最終閲覧日: 2022-09-01).

[25] 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#GUC-MAX-PARALLEL-WORKERS-PER-GATHER, (最終閲覧日: 2022-09-01).

[26] PostgreSQL グローバル開発グループ:「15.1. パラレルクエリはどのように動くのか」, PostgreSQL 12.4 文書, 2020, https://www.postgresql.jp/document/10/html/how-parallel-query-works.html, (最終閲覧日: 2022-09-01).

[27] 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#PARALLEL-LABELING, (最終閲覧日: 2022-09-01).

[28] PostGIS Project: PostGIS ホームページ, 2022, http://postgis.net, (最終閲覧日: 2022-10-28).

補遺A

SQL 1 tmq_kisoテーブルの定義

```
 1  CREATE TABLE tmq_kiso (
 2      frame_id  CHARACTER VARYING(68)  PRIMARY KEY,
 3      object    CHARACTER VARYING(68)  NOT NULL,
 4      ra        CHARACTER VARYING(68)  NOT NULL,
 5      dec       CHARACTER VARYING(68)  NOT NULL,
 6      date_obs  CHARACTER VARYING(68)  NOT NULL,
 7      time_obs  CHARACTER VARYING(68)  NOT NULL,
 8      exptime   NUMERIC(9,6)           NOT NULL,
 9      exp_id    INTEGER                NOT NULL,
10      lonpole   NUMERIC(9,6)           NOT NULL,
11      crval1    NUMERIC                NOT NULL,
12      crval2    NUMERIC                NOT NULL,
13      crpix1    NUMERIC                NOT NULL,
14      crpix2    NUMERIC                NOT NULL,
15      cd1_1     NUMERIC                NOT NULL,
16      cd1_2     NUMERIC                NOT NULL,
17      cd2_1     NUMERIC                NOT NULL,
18      cd2_2     NUMERIC                NOT NULL,
19      ap_order  SMALLINT,
20      ap_0_0    NUMERIC,
21      ap_0_1    NUMERIC,
22      ap_0_2    NUMERIC,
23      ap_1_0    NUMERIC,
24      ap_1_1    NUMERIC,
25      ap_2_0    NUMERIC,
26      bp_order  SMALLINT,
27      bp_0_0    NUMERIC,
28      bp_0_1    NUMERIC,
29      bp_0_2    NUMERIC,
30      bp_1_0    NUMERIC,
31      bp_1_1    NUMERIC,
32      bp_2_0    NUMERIC
33  );
34  CREATE INDEX ON tmq_kiso (date_obs);
35  CREATE INDEX ON tmq_kiso (exp_id);
36  CREATE INDEX ON tmq_kiso (crval1,crval2);
37  CREATE INDEX ON tmq_kiso (crpix1,crpix2);
38  CREATE INDEX ON tmq_kiso (cd1_1,cd1_2,cd2_1,cd2_2);
39  CREATE INDEX tmq_kiso_xyz_idx ON tmq_kiso (
40      feq2x(fra2deg(ra),fdec2deg(dec)),
41      feq2y(fra2deg(ra),fdec2deg(dec)),
42      feq2z(fdec2deg(dec))
43  );
```

SQL 2 filemng_tmqテーブルの定義

```
1  CREATE TABLE filemng_tmq (
2      frame_id     CHARACTER VARYING(30)  PRIMARY KEY,
3      date_obs     CHARACTER VARYING(12)  NOT NULL,
4      fits_path    CHARACTER VARYING(80),
5      fits_md5sum  CHARACTER VARYING(32),
6      publictime   CHARACTER VARYING(20),
7      publicflag   CHARACTER VARYING(1)
8  )
9  CREATE INDEX ON filemng_tmq (publictime);
10 CREATE INDEX ON filemng_tmq (publicflag);
```

SQL 3 tmq_smoka ビューの定義

```
CREATE VIEW tmq_smoka AS
    SELECT
        a.frame_id,
        a.object,
        a.ra,
        a.dec,
        feq2x(fra2deg(a.ra),fdec2deg(a.dec)) AS x,
        feq2y(fra2deg(a.ra),fdec2deg(a.dec)) AS y,
        feq2z(fdec2deg(a.dec)) AS z,
        a.date_obs,
        a.time_obs,
        a.exptime,
        a.exp_id,
        a.lonpole,
        a.crval1,
        a.crval2,
        a.crpix1,
        a.crpix2,
        a.cd1_1,
        a.cd1_2,
        a.cd2_1,
        a.cd2_2,
        a.ap_order,
        a.ap_0_0,
        a.ap_0_1,
        a.ap_0_2,
        a.ap_1_0,
        a.ap_1_1,
        a.ap_2_0,
        a.bp_order,
        a.bp_0_0,
        a.bp_0_1,
        a.bp_0_2,
        a.bp_1_0,
        a.bp_1_1,
        a.bp_2_0,
        b.publictime,
        b.publicflag
    FROM
        tmq_kiso AS a INNER JOIN filemng_tmq AS b ON a.frame_id = b.frame_id
    WHERE (
        ((b.publictime) <= TO_CHAR(TIMEZONE('UTC', CURRENT_TIMESTAMP), 'yyyy-mm-dd'))
    AND
        ((b.publicflag) = 'O')
    )
;
```

SQL 4 ピンポイント検索のSQLクエリ

```
SELECT
     frame_id,
     ra,
     dec,
     date_obs,
     time_obs,
     TO_CHAR(exptime,'999.99'),
     exp_id,
     crval1,
     crval2,
     crpix1,
     crpix2,
     cd1_1,
     cd1_2,
     cd2_1,
     cd2_2,
     ap_0_0,
     ap_0_1,
     ap_0_2,
     ap_1_0,
     ap_1_1,
     ap_2_0,
     bp_0_0,
     bp_0_1,
     bp_0_2,
     bp_1_0,
     bp_1_1,
     bp_2_0
FROM
     tmq_smoka
WHERE
     crval1 >= 9.664629665561543 AND
     crval1 <= 11.70470366777179 AND
     crval2 >= 40.50208333333333 AND
     crval2 <= 42.03541666666666
ORDER BY exp_id;
```

SQL 5 ラフ検索のSQLクエリ

```
SELECT *
FROM (
     SELECT
          DISTINCT ON (exp_id) exp_id,
          ra,
          dec,
          date_obs,
          time_obs,
          exptime
     FROM
          fconesearchtmqsmoka(83.82204166666668,-5.3910833333333334, 360)
     WHERE
          crval1 >= 76.79078355955791 AND
          crval1 <= 90.85329977377545 AND
          crval2 >= -12.391083333333334 AND
          crval2 <= 1.6089166666666666
) subA
ORDER BY exp_id;
```

SQL 6 カレンダー検索のSQLクエリ

```
SELECT
DISTINCT(exp_id),
     ra,
     dec,
     date_obs,
     time_obs,
     TO_CHAR(exptime,'999.99')
FROM
     tmq_smoka
WHERE
     date_obs = '2020-01-02'
ORDER BY time_obs;
```

SQL 7 fconesearchtmqsmokaの定義

```
CREATE FUNCTION fconesearchtmqsmoka (NUMERIC,NUMERIC,NUMERIC)
RETURNS SETOF tconesearchtmqsmoka AS $$
    SELECT
        box.frame_id,
        box.ra,
        box.dec,
        box.date_obs,
        box.time_obs,
        box.exptime,
        box.exp_id,
        box.crval1,
        box.crval2,
        box.distance
    FROM (
        SELECT
            frame_id,
            ra,
            dec,
            date_obs,
            time_obs,
            exptime,
            exp_id,
            crval1,
            crval2,
            fdistancearcminxyz (
                feq2x($1,$2),
                feq2y($1,$2),
                feq2z($2),
                x,
                y,
                z
            ) AS distance
        FROM
            tmq_smoka
        WHERE
            (x BETWEEN feq2x($1,$2)-farcmin2rad($3) AND feq2x($1,$2)+farcmin2rad($3))
                AND
            (y BETWEEN feq2y($1,$2)-farcmin2rad($3) AND feq2y($1,$2)+farcmin2rad($3))
                AND
            (z BETWEEN feq2z($2)-farcmin2rad($3) AND feq2z($2)+farcmin2rad($3))
    ) AS box
    WHERE
        box.distance <= $3;
$$ IMMUTABLE LANGUAGE SQL;
```