情報処理学会論文誌 Vol.55 No.7 1637–1644 (July 2014)

RDBMSにおけるクエリ実行計画情報を用いたストレージ階層制御による高速化手法

松沢 敬一[1,a)] 林 真一[1] 大谷 俊雄[2] 岩嵜 正明[1]

受付日 2013年10月30日, 採録日 2014年4月4日

概要：ストレージ階層制御は，複数の記憶デバイスを使い分け，高頻度でアクセスされる一部のデータをSolid State Drive（SSD）のような高性能な記憶デバイスに配置することでアプリケーションの実行速度を向上する技術である．ストレージ階層制御の効果は，高頻度でアクセスされるデータをいかに適切に選択し，SSD に配置するかで決定される．本論文では，Relational Database Management System（RDBMS）を対象アプリケーションとし，そのクエリ処理を高速化するストレージ階層制御方式 Pre-query Storage Tiering（PST）を提案する．PST では，従来方式の 1 つである論理ボリューム方式に加え，RDBMS のクエリ実行計画情報に基づいてクエリ実行開始時にデバイス間でデータ再配置を行い，RDBMS が I/O 要求を行う時点までに要求先データが SSD に格納されるようにする．PST を Linux カーネルと MySQL に適用して評価した結果，TPC-H ベンチマークの実行において PST は従来の論理ボリューム方式に比べて HDD への I/O 要求回数を削減させ，77%の実行時間でクエリ処理を完了できた．

キーワード：階層ストレージ管理，SSD，RDBMS，最適化

Pre-query Storage Tiering by Exploring RDBMS Query Execution Plan

Keiichi Matsuzawa[1,a)] Shinichi Hayashi[1] Toshio Otani[2] Masaaki Iwasaki[1]

Received: October 30, 2013, Accepted: April 4, 2014

Abstract: Storage tiering is a technique to move data onto appropriate storage devices among variety of storage devices. It allocates the frequently accessed data onto high performance devices like Solid State Drive (SSD) and improves the average I/O response time for the storage system. The effectiveness of tiering depends on how to select such data and relocate them onto SSDs. In this paper, we propose Pre-query Storage Tiering (PST) to improve the query performance of Relational Database Management System (RDBMS). PST manages a logical volume consisting of HDDs and SSDs, predicts data area where RDBMS will access in query execution, and relocates such data onto SSDs before query execution by using query execution plan. The experiments of PST implemented on Linux kernel and MySQL show that PST can reduce I/O requests to HDDs and provide an improvement of up to 77% in query processing time on TPC-H benchmark as compared to the previous storage tiering methods.

Keywords: hierarchical storage management, SSD, RDBMS, optimization

1. はじめに

コンピュータが処理対象とするデータ容量は年々増加している．データを格納する記憶デバイスとして広く用いられている HDD の記憶容量は今後も増加を続けると予測されており [1]，HDD は今後も依然としてデータの中心的な記憶デバイスであり続ける．しかし，HDD の I/O 性能

[1] 株式会社日立製作所横浜研究所
Yokohama Research Laboratory, Hitachi Ltd., Yokohama, Kanagawa 244–0817, Japan
[2] 株式会社日立製作所 IT プラットフォーム事業本部
IT Platform Division Group, Hitachi Ltd., Yokohama, Kanagawa 244–0817, Japan
[a)] keiichi.matsuzawa.kd@hitachi.com

の向上ペースは，HDD の記憶容量の増加ペースや CPU・ネットワーク等，他のコンピュータの構成要素の性能向上ペースに比べ遅く，データ容量に対して I/O 性能不足が生じている [2], [3].

近年では，HDD に代わり記憶デバイスとしてフラッシュメモリを用いた Solid State Drive（SSD）の普及が進んでいる．SSD は I/O 性能で HDD に比べ優れる反面，1 台あたりの記憶容量は HDD より小さく，ビット単価が高い [4]. そのため，容量・ビット単価と性能において HDD と SSD はトレードオフの関係にある．

このように特性の異なる記憶デバイスを使い分ける技術として，データをアクセスパターンに適した記憶デバイスに配置するストレージ階層制御がある．HDD と SSD を用いるストレージ階層制御では，高頻度でアクセスされる一部のデータを SSD に配置し，残りのデータを HDD に配置することで，HDD の特性である大容量・低コストを維持しつつ，HDD よりも平均 I/O 性能の優れたストレージを実現できる．

ストレージ階層制御では，データをどのように記憶デバイスに配置するかによって各デバイスに実行される I/O 要求の回数が変わるため，同容量の SSD を用いた場合でもデータ配置の方式によって得られる性能は変動する．

本論文では，Relational Database Management System（RDBMS）におけるクエリ処理の高速化を目的としたストレージ階層制御方式である Pre-query Storage Tiering（PST）を提案する．PST はボリューム管理機能で複数記憶デバイスの使い分けを行う論理ボリューム方式に加え，RDBMS から取得したクエリ実行処理情報を用いて RDBMS が I/O を要求する前にデータ再配置を行い，クエリ処理を高速化する．また，本論文では RDBMS の一実装である MySQL を対象として PST を実装し，性能評価を行う．

本論文の構成は以下のとおりである．まず，2 章で従来のストレージ階層制御方式とその課題を説明し，続く 3 章で従来研究に対する PST の位置づけを述べる．4 章では PST の設計を示し，5 章では PST の性能評価を行う．6 章で複数の RDBMS を対象として PST の適用可否と必要変更量について論ずる．最後に 7 章で本論文のまとめを行う．

2. 従来方式とその課題

RDBMS を対象としたストレージ階層制御の先行研究を**表 1** に示す．この表は，階層制御の実装レイヤ，再配置に利用するヒント情報，データ再配置を行うタイミングの観点で分類している．

2.1 RDBMS による直接階層制御

ストレージ階層制御方式の 1 つに，RDBMS が直接 HDD と SSD を管理し，それぞれにデータを格納する直接階層制御方式がある．この方式に分類できる先行研究としては，HDD のリードキャッシュとして SSD を用いる SSD Bufferpool [5] や，更新データのライトバックキャッシュとして SSD を用いる In-page Logging [6] がある．これらは，RDBMS が I/O 要求の特性に応じてデータを SSD や HDD に配置する方式である．

表 1 ストレージ階層制御方式の分類
Table 1 Storage tiering approaches.

階層制御の実行レイヤ	利用するヒント情報	再配置のタイミング	
		先行再配置	I/O 要求時再配置
RDBMS	RDBMS		SSD Bufferpool In-page Logging
論理ボリューム	RDBMS	PST	hStorage-DB CLIC
論理ボリューム	論理ボリューム	KNOWAC LAM	bcache FlashCache

直接階層制御方式では，対象の RDBMS 固有の情報を用いてデータ配置を最適化できる．たとえば In-page Logging では RDBMS が I/O 要求を行う記憶デバイス上の位置を考慮して SSD に配置するデータを決定する．そのため RDBMS 固有の情報を利用できない論理ボリューム方式に比べ，高い性能改善効果が期待できる．一方，直接階層制御の実装に必要な変更箇所は個々の RDBMS に依存しており，他の RDBMS への流用が難しく汎用性が低い．

2.2 論理ボリューム方式

論理ボリューム方式は，RDBMS がデータを格納するボリュームの管理機能において HDD と SSD を使い分ける方式である．

論理ボリューム方式の実装としては，HDD に対して SSD をキャッシュとして用いる機能を備えた論理ボリューム構築を作成する bcache [7] や FlashCache [8] がある．bcache は更新データのライトキャッシュとして SSD を利用し，FlashCache はリードキャッシュとして SSD を利用する．いずれも，I/O 要求時にデータを HDD や SSD に配置する方式である．

論理ボリューム方式において I/O 要求に先行してデータ再配置を行う研究として，KNOWAC [9] や Lookahead Data Migration（LAM）[10] がある．KNOWAC や LAM は，論理ボリュームに対する I/O 要求の履歴を分析し，以後 I/O 要求が行われると予測した論理ボリューム上の領域のデータを事前に SSD に配置しておく．論理ボリューム方式はボリュームの管理機能で階層制御を行うため，RDBMS の変更が不要という利点がある．反面，ボリューム管理機能で得られる情報のみを用いてデータ再配置を行うため，ワークロードが時間経過にともない変化するケースでは I/O 要求に適したデータの配置が遅れ，高い性能向上効果を得にくい [11].

2.3 RDBMS のヒント情報と論理ボリュームの併用方式

直接階層管理と論理ボリューム方式の短所を補うため，論理ボリューム方式のデータ再配置に RDBMS が生成する

ヒント情報を用いる先行研究もある．

CLIC [12] は RDBMS が I/O 要求にヒント情報としてテーブル名を付与するインタフェースを提供する．ボリューム管理機能はこのヒント情報ごとに過去の I/O 要求の特性を分析し，SSD に再配置するデータを決定する．

hStorage-DB [13] は RDBMS が生成するクエリ実行計画に基づき，I/O 要求にヒント情報として I/O 優先度情報を付与するインタフェースを提供する．ボリューム管理機能は，優先度の高い I/O 要求が参照する論理ボリューム上の領域を SSD に再配置し性能を向上する．

CLIC と hStorage-DB はいずれも I/O 要求に対してヒント情報を付与する手法であり，再配置は I/O 要求時に行われる．

3. 提案する PST の位置づけ

本論文は，RDBMS のクエリ実行計画情報を用いて，クエリ処理中に RDBMS が行う I/O 要求に先行してデータ再配置を行う Pre-query Storage Tiering（PST）を提案する．PST は論理ボリュームに対し，RDBMS からヒント情報を付与する点では先行研究の CLIC や hStorage-DB と類似する．しかし，これらの先行研究が I/O 要求時にデータを再配置するのに対し，PST では I/O 要求に先行してデータを再配置する点が異なる．

RDBMS のクエリ処理では，クエリ実行計画を作成した後，計画に基づいてクエリを実行し，その実行過程で論理ボリュームに I/O 要求を行う．PST ではクエリ実行計画作成時点でデータを再配置しておくことで，I/O 要求時点までに対象データを SSD に再配置させることができ，I/O 要求の応答時間を短縮することができる．

また，PST はヒント情報として RDBMS が標準的に備えるクエリ実行計画情報を用いるため，ヒント情報を用いる他の先行研究のように RDBMS の大幅な変更を行う必要がない．

4. PST の設計と実装

本章では，PST の構成要素および動作と，Linux カーネルおよび MySQL における PST の実装方法について説明する．

4.1 PST の構成

図 1 に PST の構成を示す．PST の構成要素は，RDBMS，ボリューム管理機能，最適配置計算プログラムからなる．

ボリューム管理機能は，HDD と SSD から単一の論理ボリュームを作成し，RDBMS はその論理ボリューム上に複数のテーブルやインデックスを格納する．最適配置プログラムは，RDBMS がクエリ処理において生成するクエリ実行計画情報を用いて，クエリ実行中に参照されるテーブルやインデックスを特定する．それらのテーブルやインデックスを格納している論理ボリューム上の領域は，クエリ実行中に I/O 要求が行われると予測できるので，最適配置計算プログラムは当該領域に格納されたデータを SSD に再配置するようボリューム管理機能に指示する．さらに，最適配置計算プログラムはクエリ実行計画情報に加えて，RDBMS が出力する性能統計情報やボリューム管理機能が出力する I/O 統計情報を用いて，I/O 要求が行われる論理ボリューム上の領域をより細粒度で予測する．

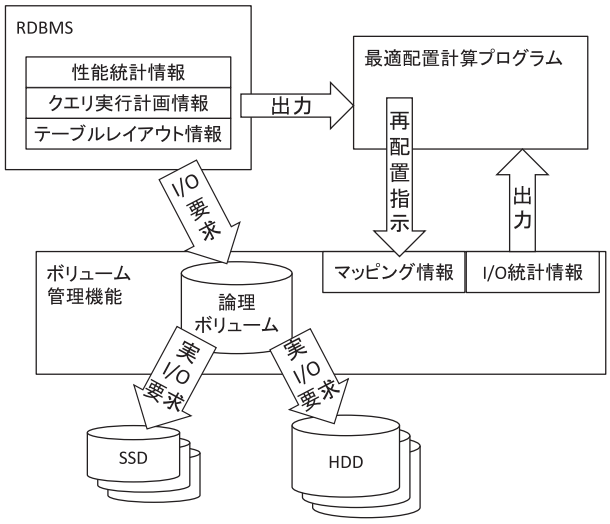


図 1 PST の全体構成
Fig. 1 PST architecture.

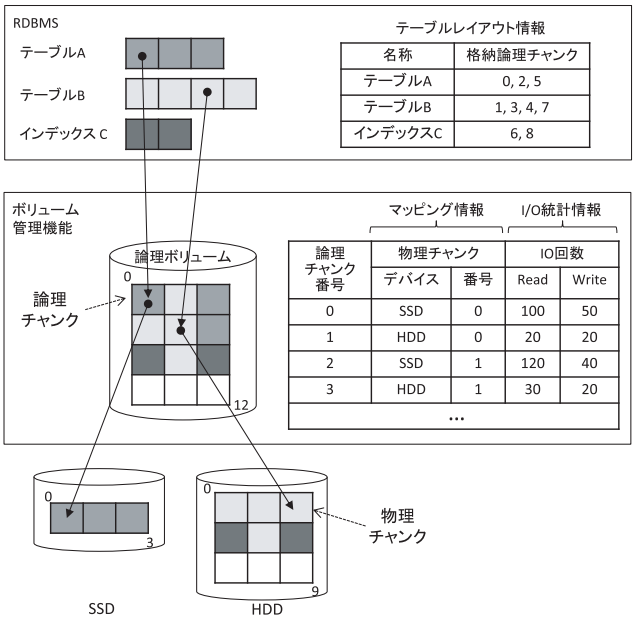


図 2 PST におけるデータ構造
Fig. 2 Data structure in PST.

4.2 ボリューム管理機能と PST のデータ構造

ボリューム管理機能は，RDBMS がデータを格納するための論理ボリュームを作成する．**図 2** に PST における RDBMS と記憶デバイス間のデータ構造について示す．PST では，論理ボリュームおよび SSD・HDD の領域を固定長チャンクの集合として管理し，論理ボリュームを構成す

るチャンクを論理チャンク，SSD・HDD を構成するチャンクを物理チャンクと呼ぶ．論理チャンクと物理チャンクはそれぞれ 1 対 1 で対応づけられ，その対応関係はボリューム管理機能中のマッピング情報に保持する．RDBMS が論理ボリュームに I/O 要求を行うと，ボリューム管理機能は I/O 要求がなされた論理チャンクに対し，マッピング情報を参照して対応する物理チャンクを求め，論理チャンクへの I/O 要求を物理チャンクへの I/O 要求（以後，実 I/O 要求と呼ぶ）に変換し，SSD・HDD に対して実 I/O 要求を行う．各論理チャンクを異なる記憶デバイスの物理チャンクに対応づけることで，各論理チャンクはより適切な性能特性を持つことができる．また，ボリューム管理機能は，論理チャンクごとに I/O 回数等の I/O 統計情報を集計する．この統計情報は最適配置計算プログラムによって参照され，最適配置計算で利用される．

RDBMS が管理するテーブルやインデックスは，それぞれ論理ボリューム上の複数の論理チャンクに格納され，格納先の論理チャンクは RDBMS 内のテーブルレイアウト情報で管理される．テーブルレイアウト情報とマッピング情報を両方参照することでテーブルやインデックスが格納された物理チャンクを特定できるため，チャンク単位でのデータ再配置が可能となる．

4.2.1 データ再配置の手順

ボリューム管理機能は，最適配置計算プログラムの指示に応じてデータ再配置を行う．データの再配置とは，論理チャンクの対応づけを異なる性能特性の物理チャンクへ変える処理であり，変更前後の物理チャンク間のデータコピーとマッピング情報変更の 2 ステップで行われる．最適配置計算プログラムはボリューム管理機能に論理チャンクと新たに対応づける変更先物理チャンクを伝えて再配置を指示する．ボリューム管理機能が再配置指示を受け取ると，ボリューム管理機能は変更元物理チャンクの格納データを変更先物理チャンクにコピーし，その後マッピング情報における論理チャンクと対応づけられる物理チャンクを，変更先物理チャンクに変更する．データ再配置中の論理チャンクに対する I/O 要求は，再配置が完了するまで遅延されるため，RDBMS が再配置中の論理チャンクに I/O 要求を実行してもデータの不整合は生じない．

4.3 PST で用いる RDBMS のヒント情報

PST では最適配置計算プログラムが以後 I/O 要求が行われる論理チャンクを予測するため，RDBMS が出力するクエリ実行計画情報および性能統計情報を用いる．

クエリ実行計画情報は，RDBMS がクエリ実行中に行う内部処理を表したもので，実行中に参照するテーブル名やインデックス名を含んでいる．PST ではこのテーブル名やインデックス名をヒント情報として用いる．

性能統計情報は，過去の RDBMS のクエリ処理におけるテーブル参照・更新にともなう I/O 回数や I/O 応答時間を含む．この性能統計情報を用いることで，次節に述べるように，以後 I/O 要求が行われる論理チャンクの予測精度を高めることができる．

4.4 最適配置計算プログラムの動作

最適配置計算プログラムは，RDBMS が出力するヒント情報とボリューム管理機能が出力する情報を元に，RDBMS のクエリ処理がより高速になるデータ配置を計算し，ボリューム管理機能に再配置指示を行う．

最適配置計算プログラムは，まずクエリ実行計画情報からクエリ処理中に参照されるテーブルやインデックスを求める．さらに過去の RDBMS の性能統計情報を用いて，以後の I/O 要求の頻度が特に高いテーブルやインデックスを予測する．次に，最適配置計算プログラムはテーブルレイアウト情報を参照して，I/O 要求頻度が高いと予測したテーブルやインデックスが格納された論理チャンクを特定する．さらに，最適配置計算プログラムはボリューム管理機能の I/O 統計情報を取得し，論理チャンク単位で I/O 要求の頻度の高さを予測する．

最適配置計算プログラムはここまでの手順で以後 I/O 要求頻度が高くなる論理チャンクを予測できるので，予測した I/O 要求頻度が高い順に論理チャンクを SSD の容量分まで SSD の物理チャンクに再配置し，残りの論理チャンクを HDD に再配置するようボリューム管理機能に指示する．

5. TPC-H を用いた性能評価

5.1 評価方法

本実験では，PST の有効性を検証するため，性能ベンチマーク対象として TPC-H [14] を用いた実測を行った．TPC-H は意思決定のためのデータ分析を模したベンチマークプログラムであり，大容量のテーブルを参照するためディスクアクセス性能が顕著に実行時間に影響する．

5.1.1 測定対象クエリの選出

TPC-H は 22 種のクエリの実行時間を測定するベンチマークプログラムである．しかし本実験ではストレージ階層制御の効果を明確にするため，特にディスクアクセス性能がボトルネックとなるクエリ 9 種（Q3，Q4，Q5，Q7，Q8，Q9，Q10，Q20，Q21）を選択し，各クエリを順次実行した合計時間を測定対象とした．これらのクエリは，事前に MySQL の全データを HDD に格納した場合と SSD に格納した場合におけるクエリの実行時間を測定し，両測定の実行時間比が大きいものを選出したものである．

5.1.2 比較対象とする階層制御方式

本評価実験では比較対象となるストレージ階層制御方式として固定配置方式（FST）および FlashCache を用いて，PST との比較評価を行った．

FST と PST はいずれも同じ論理ボリューム上にデータ

を配置するが，両者は再配置に用いるヒント情報と再配置のタイミングが異なる．両方式ともに，事前にすべての論理チャンクを HDD の物理チャンクに対応づけた状態で一度 TPC-H を実行し，各チャンクに実行された全クエリの合計 I/O 要求回数を採取して，再配置のヒント情報として高頻度で I/O 要求が出される論理チャンクの予測に用いる．FST の場合，最適配置計算プログラムは RDBMS のヒント情報は利用せず，採取した I/O 要求回数が大きい順で論理チャンクを SSD に対応づけ，残りを HDD に対応づける．また，FST はクエリの開始を認識しないため，TPC-H 実行開始時に一度だけデータ再配置を行う．一方 PST は FST 同様に I/O 要求回数をヒント情報として用いるが，TPC-H の各クエリ処理前に EXPLAIN 句を実行し，得られたクエリ実行計画情報を併用してクエリごとにデータ再配置を行う点が異なる．PST では MySQL によるクエリ実行と並列して最適配置計算プログラムがデータ再配置を実行する．そのため，測定結果における実行時間にはデータ再配置処理も含む．

FlashCache は，HDD 上のデータを SSD 上でキャッシュする機能を提供する．FlashCache は HDD への I/O 要求を常時監視し，I/O 要求時点において LRU 順でデータを SSD に再配置する．

5.2 実装

本実験では，Linux カーネルが備える Device Mapper [15] を用いて，FST および PST が利用するボリューム管理機能を実装した．このボリューム管理機能は ioctl システムコールを備えており，最適配置計算プログラムから I/O 統計情報の取得や再配置指示が可能である．

対象 RDBMS としては MySQL を用いた．MySQL のストレージエンジンである InnoDB [16] は論理ボリュームを 1 MB 固定長領域の集合として管理し，複数のテーブルやインデックスをこの固定長 1 MB 単位で配置する．そこで論理チャンクの大きさを同じ 1 MB にすることで，InnoDB と論理ボリュームのデータ配置をアラインさせた．

PST で用いる RDBMS のヒント情報のうち，性能統計情報とクエリ実行計画情報は MySQL の標準機能である performance_schema スキーマと EXPLAIN クエリを用いて出力した．テーブルレイアウト情報は MySQL が標準で出力機能を有していないため，InnoDB のデバッグ情報出力機能に 70 ステップの改造を加え，テーブルレイアウト情報を追加出力するようにした．

最適配置計算プログラムは Perl で作成した．この最適配置計算プログラムは MySQL とは別プロセスとして動作し，MySQL がクエリ実行計画情報を出力すると，本プログラムは MySQL のクエリ実行と並行して最適配置計算およびボリューム管理機能への再配置指示を行う．

表 2 TPC-H 実験環境
Table 2 TPC-H test environment.

項目	内容
CPU	Xeon 5140 (2core, 2.33GHz) ×1
メモリ	DDR3 12GB (うち 1GB のみ使用)
HDD	HGST HDS723020BLA642×3 台 (2TB, 64MB cache, 7200rpm) RAID0 で使用(ストライプ幅 64KB)
SSD	Intel SSD 320 Series×1 台 (300GB)
Operating System	Fedora 15 (x86_64)
Linux kernel	2.6.43.8-1.fc15.x86_64
IO scheduler	deadline (HDD), noop (SSD)
MySQL	5.6.15 (レイアウト情報出力パッチ済)
TPC-H version	DBT3 version 1.9
TPC-H parameter	Scale factor = 1
FlashCache	version 0.3

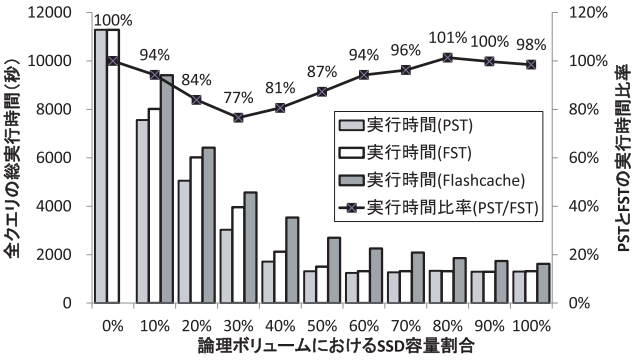


図 3 全クエリ合計実行時間
Fig. 3 Total execution time of queries.

5.3 測定環境

本実験の測定環境を**表 2** に示す．記憶デバイスには HDD を 3 台 RAID-0 でストライプ構成を組んだものと，SSD 1 台を用いた．

本測定におけるデータセットのサイズは，TPC-H の Scale factor パラメータを 1 に設定して得られたテーブルデータ 1.22 GB とインデックス 1.49 GB である．階層ストレージ管理の方式の違いが明確となるようディスク I/O 待ちがボトルネックとなる環境を模擬するため，OS が使用するメモリ容量をデータ容量より小さい 1 GB に制限して MySQL や OS のディスクキャッシュの効果を小さくした．

5.4 測定結果と分析

5.4.1 TPC-H 実行時間比較

論理ボリュームに占める SSD の容量割合を 10%刻みで変えながら，各方式において 9 種のクエリの合計実行時間を複数回測定した平均値を**図 3** に示す．ただし FlashCache はキャッシュ容量 0%では動作しないため，SSD 容量割合 0%の測定を含まない．各測定では TPC-H に同一の乱数シード値を与えており，クエリの実行順は同じである．

各方式とも SSD 容量割合が増加するに従ってクエリ実行時間は減少した．また，I/O 要求に先行してデータ再配置を行う FST や PST の方が，I/O 要求時にデータ再配置を行う FlashCache よりも良い結果を得た．

FST と PST を比較すると，SSD 容量割合が少ない場合

には PST の実行時間が FST を大きく下回り，PST は最短で FST の 77%の時間で処理を終えた．一方，SSD 容量割合が増加すると PST と FST の実行時間は同程度となり，PST の処理時間が FST より長くなる測定ケースもあった．この要因については，後述の実験でさらに分析する．

5.4.2 HDD への実 I/O 要求変換割合評価

PST が FST よりも SSD を有効活用できていることを検証するため，クエリ実行中に実行された論理ボリュームへの I/O 要求のうち，ボリューム管理機能により HDD への実 I/O 要求に変換された回数を比較した．HDD は SSD よりも I/O 要求への応答時間が長いため，この回数が少ないほど平均 I/O 時間が短くなる．

図 4 に全クエリ処理における HDD への実 I/O 要求回数を示す．PST と FST の両方式とも SSD 容量割合が増加するにつれ，HDD への実 I/O 変換回数が減少した．ここで，クエリの合計実行時間比較で PST が FST より短かった SSD 容量割合 50%以下のケースにおいて，PST は FST より HDD への実 I/O 要求回数が少なく，限られた SSD 容量をより効率的に活用できていることが分かる．クエリの合計実行時間で顕著な差が見られない SSD 容量が 60%以上の測定ケースにおいては，PST と FST はともに HDD への実 I/O 要求回数は全 I/O 要求の 1%以下であり，両者の差はわずかとなる．このため，PST が FST と比べ I/O 性能の改善効果を得られないことが分かる．

5.4.3 データ再配置のオーバヘッド評価

本項では PST がクエリ間に実行するデータ再配置のオーバヘッドを検証する．TPC-H のクエリ実行中，データ再配置のために実行した HDD への I/O 要求回数を**図 5** に示す．論理ボリュームにおける SSD 容量割合が 30%以下と少ない場合，図 4 に示す通りクエリ実行中の HDD への実 I/O 要求回数は 30 万回以上である．そのため PST 再配置で行う HDD への I/O 回数は 2 万回未満と相対的に少ないため，性能改善効果が再配置のオーバヘッド影響を上回ったと考えられる．しかし SSD 容量割合が 70%を超えるとクエリ実行中の HDD への実 I/O 要求回数は 2,000 回以下となり，相対的に再配置のための I/O 回数の影響が増加する．そのため PST の処理オーバヘッドが増加し，PST が FST よりもクエリ処理時間が長くなったと考えられる．

5.4.4 クエリごとの性能改善効果分析

クエリの特性による PST の適性を調べるため，クエリごとに PST と FST 両方式における性能向上効果を比較した．**図 6** は，両方式の性能差が小さいクエリ（Q9）と大きいクエリ（Q5）の実行時間を示す．

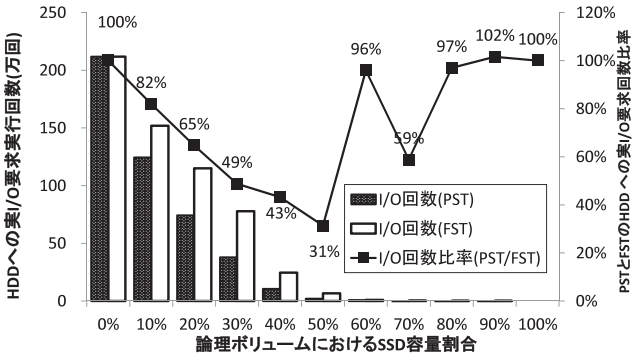


図 4 HDD への実 I/O 要求回数
Fig. 4 Number of I/Os issued to HDDs.

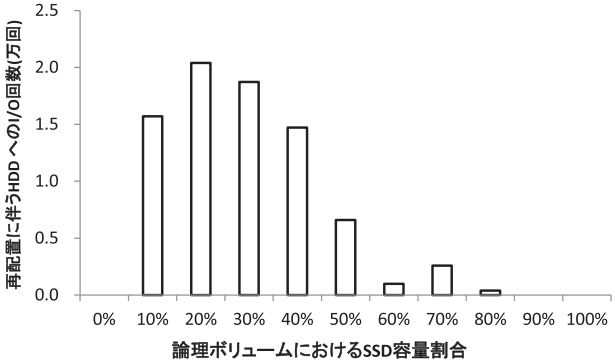
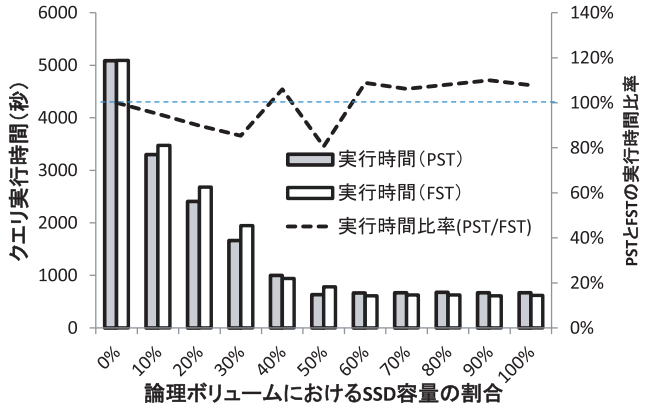
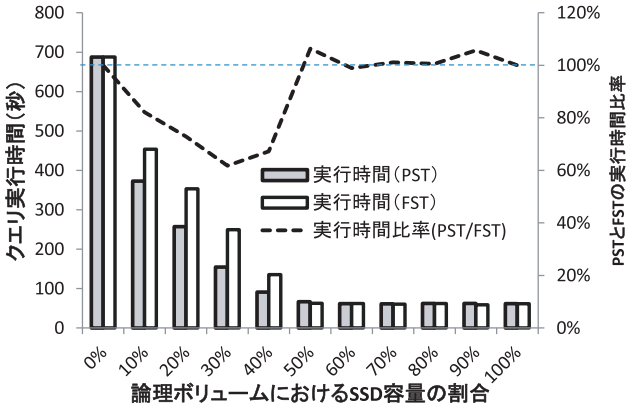


図 5 データ再配置のための I/O 要求回数
Fig. 5 I/O requests to relocate data.



(a) クエリ Q9 の実行時間



(b) クエリ Q5 の実行時間

図 6 クエリの違いによる両方式の効果比較
Fig. 6 Query and storage tiering methodology correlations.

クエリ Q9 では PST は FST に比べ最良でも実行時間比率 80%（SSD 容量割合 50%）と顕著な性能改善効果を得られず，SSD 容量割合 40%および 60%以上においては逆に FST より実行時間が長くなった．一方クエリ Q5 では，PST の実行時間が最良で FST の 62%（SSD 容量割合 30%）と SSD 容量割合が少ないケースで大幅な性能改善効果を得た．

Q9 と Q5 のクエリ実行計画を比較すると，Q9 では TPC-H のデータセット容量の 29%を占める lineitem テーブルを参照している．この lineitem テーブルは多くのクエリで参照されており，その結果 5.1.2 項で取得した全クエリ合計の I/O 要求回数と，Q9 の I/O 要求回数の傾向が近く，FST で十分に Q9 の I/O 要求に適したデータ配置ができていたと考えられる．一方，Q5 の実行時間の多くを占める lineitem テーブルと order テーブルの 2 つのインデックスは，3 種のクエリでしか参照されない．これらのテーブル・インデックスは 9 つのクエリの合計 I/O 要求回数で比較すると他のテーブル・インデックスに比べて回数が少ないため，FST ではこれらのテーブル・インデックスの格納された論理チャンクは SSD に配置されない．一方 PST では，Q5 の実行に際しこれらのテーブル・インデックスを適切に選択して論理チャンクを SSD に再配置したため，大きな性能改善効果を得られたと考えられる．

6. PST 適用における RDBMS の必要変更量

PST を RDBMS に適用するために必要な RDBMS のソースコード変更量を評価するため，**表 3** にボリューム上にテーブルを格納する RDBMS 実装として MySQL，Oracle Database，SQLite を例にあげ，PST の適用可否を示す．

PST は階層制御の実装レイヤがボリューム管理機能となるため，RDBMS は最適配置計算プログラムが利用するヒント情報を出力するだけで利用できる．

MySQL は前述のとおり，最適配置計算プログラムが用いる情報のうちクエリ実行計画および性能統計情報の出力機能を標準で備えている．しかし MySQL はテーブルレイアウト情報の出力機能を有しないため，MySQL を約 70 ステップ変更することでテーブルレイアウト情報の出力機能を追加し，PST を適用できた．Oracle Database は標準機能でクエリ実行計画情報，性能統計情報，テーブルレイアウト情報 [17] をいずれも出力可能である．そのため，Oracle Database には変更を加えることなく PST を適用できる．

表 3 RDBMS の PST 適用可否と必要変更量
Table 3 Functions and modifications required to adopt PST.

RDBMS 実装	クエリ実行計画情報出力	I/O 統計情報出力	テーブルレイアウト情報出力	PST 適用可否
MySQL version 5.6	あり	あり	なし（70step）	可
Oracle Database 12c	あり	あり	あり	可
SQLite version 3.8	あり	なし	なし（40step）	可

SQLite [18] は MySQL 同様にテーブルレイアウト情報の出力機能を標準で有さないため，約 40 ステップの変更によりテーブルレイアウト情報の出力機能を追加することで，PST の適用が可能となる．ただし，SQLite は性能統計情報を持たないため，高頻度で I/O 要求がなされる論理チャンク予測に性能統計情報を利用できず，性能向上効果が他の RDBMS より小さくなると推測する．

このように，PST が利用する RDBMS の情報の多くは RDBMS が標準機能で備えるものであり，不足する機能もわずかな変更を加えることで対応できることが分かった．

7. おわりに

ストレージ階層制御において効率的に性能を向上するには，アプリケーションがアクセスするデータを適切に SSD に配置することが必要である．

本論文では，従来の論理ボリューム方式に加え，RDBMS が出力するクエリ実行計画情報を用いて I/O 要求に先行してデータ再配置を行う PST を提案した．PST により，HDD に発行される実 I/O 要求が減少するため，RDBMS のクエリ処理を高速化できる．

さらに本論文では，PST を Linux カーネルおよび MySQL 上に実装し，TPC-H を用いて性能測定を行った．その結果，同一容量の SSD を使用した場合に従来手法と比較して最良で 77%の実行時間でクエリ処理を完了し，本方式の有効性を実証できた．一方，SSD の容量が十分に多い場合，従来手法と PST における性能改善効果の差は小さくなることを示した．

参考文献

[1] 堀内義章：2013 年・世界経済と HDD 業界展望，IDEMA Japan News No.112（オンライン）(2013)，入手先 ⟨http://www.idema.gr.jp/news/112/⟩（参照 2014-02-17）．
[2] Freitas, R.F., Slember, J., Sawdon, W. and Chiu, L.: GPFS Scans 10 Billion Files in 43 Minutes, IBM Research Report, RJ10484 (online) (2011), available from ⟨http://domino.watson.ibm.com/library/CyberDig.nsf/papers/4A50C2D66A1F90F7852578E3005A2034⟩ (accessed 2014-02-17).
[3] Spanjer, E.: Stop Wasting Money on $/GB and Invest on $/TBW, *Proc. 2013 Flash Memory Summit* (2013), available from ⟨http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130813_D11_Spanjer.pdf⟩ (accessed 2014-02-17).
[4] Fontana, R.E., Hetzler, S.R. and Decad, G.: Technology Roadmap Comparisons for TAPE, HDD, and NAND Flash: Implications for Data Storage Applications, *IEEE Trans. Magnetics*, Vol.48, No.5, pp.1692–1696 (2012).

[5] Canim, M., Mihaila, G.A., Bhattacharjee, B., Ross, K.A. and Lang, C.A.: SSD Bufferpool Extensions for Database Systems, *Proc. 36th International Conference on Very Large Data Bases* (*VLDB2010*), Vol.3, No.1-2, pp.1435–1446 (2010).
[6] Lee, S.W. and Moon, B.: Design of Flash Based DBMS: An In-page Logging Approach, *Proc. 2007 ACM International Conference on Management of Data* (*SIGMOD2007*), pp.55–66 (2007).
[7] bcache (online), available from ⟨http://bcache.evilpiepirate.org/⟩ (accessed 2014-02-17).
[8] FlashCache (online), available from ⟨https://github.com/facebook/flashcache⟩ (accessed 2014-02-17).
[9] He, J., Sun, X. and Thakur, R.: KNOWAC: I/O Prefetch via Accumulated Knowledge, *Proc. 2012 IEEE International Conference on Cluster Computing* (*CLUSTER '12*), pp.429–437 (2012).
[10] Zhang, G., Chiu, L., Dickey, C., Liu, L., Muench, P. and Seshadri, S.: Automated lookahead data migration in SSD-enabled multi-tiered storage systems, *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies* (*MSST '10*), pp.1–6 (2010).
[11] 大江和一，荻原一隆，野口泰生，小沢年弘：spike 領域をリアルタイムに高速ストレージに移動することが可能な階層ストレージシステムの提案，情報処理学会論文誌コンピューティングシステム，Vol.5, No.5, pp.118–127 (2012).
[12] Liu, X., Aboulnaga, A., Salem, K. and Li, X.: CLIC: CLient-Informed Caching for Storage Servers, *Proc. 7th USENIX Conference on File and Storage Technologies* (*FAST '09*), pp.297–310 (2009).
[13] Luo, T., Lee, R., Mesnier, M., Chen, F. and Zhang, X.: hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems, *Proc. VLDB Endowment*, Vol.5, No.10, pp.1076–1087 (2012).
[14] Transaction Processing Performance Council: The TPC Benchmark^{TM} H (TPC-H) (online), available from ⟨http://www.tpc.org/tpch/⟩ (accessed 2014-02-17).
[15] Device-mapper Resource Page (online), available from ⟨http://sourceware.org/dm/⟩ (accessed 2014-02-17).
[16] Sun, C.: InnoDB Internals: InnoDB File Formats and Source Code Structure, *Proc. 2009 MySQL Conference & Expo* (2009).
[17] Oracle Database 管理者ガイド 12c リリース 1 (online), available from ⟨http://docs.oracle.com/cd/E49329_01/server.121/b71301/dfiles.htm⟩ (accessed 2014-02-17).
[18] SQLite Home Page, available from ⟨http://www.sqlite.org/⟩ (accessed 2014-02-17).

松沢 敬一 （正会員）

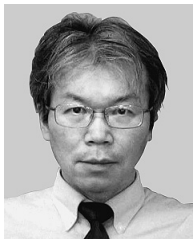
2003 年東京大学理学部情報科学科卒業．2005 年同大学大学院学際情報学府修士課程修了．同年（株）日立製作所入社．システム開発研究所を経て，現在，横浜研究所勤務．ストレージシステムに関する研究に従事．

林 真一 （正会員）

2003 年東京学芸大学教育学部情報環境科学課程教育情報科学専攻卒業．2005 年東京工業大学大学院社会理工学研究科人間行動システム専攻修士課程修了．同年（株）日立製作所入社．システム開発研究所を経て，現在，横浜研究所勤務．アプリケーション性能向上に関する研究に従事．電気学会会員．

大谷 俊雄

1999 年同志社大学経済学部卒業．同年（株）日立製作所入社．中央研究所，システム開発研究所，横浜研究所を経て現在，情報・通信システム社・IT プラットフォーム事業本部勤務．IT システムの効率化に関する研究に従事．



岩嵜 正明 （正会員）

1981 年九州工業大学電子工学科卒業．1983 年九州大学大学院総合理工学研究科修士課程修了．同年（株）日立製作所入社．中央研究所，システム開発研究所を経て，現在，横浜研究所主管研究長．並列推論マシン，メインフレーム，超並列マシン CP-PACS（SR2201），NAS 等の OS 研究開発を経て，近年はクラウド関連の研究に従事．2008 年岡山大学にて博士号取得．IEICE，IEEE CS 各会員．