データベース再編成機構を有するストレージシステム

合田和生[†] 喜連川優[†]

デバイス技術の進展からストレージシステムは急速に高機能化しており，従来はすべてサーバ上で行われていた物理的なデータ管理の一部をストレージ上で行うことが可能となりつつある．本論文は，データベース管理において不可欠なデータベース再編成機能を有する高機能ディスクアレイ，すなわち自己再編成ストレージ（SRS）と名付けたストレージシステムを提案する．SRS はデータベース再編成をその内部でオンラインに実施することにより，データベース内のデータ構造の効率性を保つことが可能であり，情報システムの設計の容易化に貢献すると期待される．また，SRS はディスクアレイ内の豊富な IO 帯域と高い IO 処理能力を有効に利用することを目的として，並列パイプライン化データ処理，物理アドレスレベル IO スケジューリング，および独自の高速ログ適用処理を実施する．本論文は SRS の設計を述べるとともに，商用 DBMS およびオープンソース DBMS を対象とした試作機による性能評価実験を示し，サーバ上で実行される従来的なデータベース再編成に比べ，性能の大幅な改善が可能であることを明らかにする．

Storage Systems with Database Reorganizing Facility

Kazuo Goda[†] and Masaru Kitsuregawa[†]

The recent evolution of device technologies has provided storage systems with powerful processors, large cache memory and high internal bandwidth. Storage systems today have potential for intelligent data management using such resources effectively. This paper proposes *Self-Reorganizing Storage* (*SRS*), a highly functional disk array which has the capability of database reorganization. SRS conducts database reorganization online in itself and keeps the structural efficiency independently of DBMS running on server systems. Parallel pipelined data processing, physical IO scheduling and high-speed log application mechanism are adopted in order to take full advantage of high internal bandwidth and IO processing power of disk storage systems. An SRS prototype is implemented using a commercial DBMS and an open-source DBMS. The experimental results reveal that significant performance improvement is achieved.

1. はじめに

ストレージ技術の潮流は大きく変化しつつある．デバイス技術の進展によりプロセッサやメモリは高性能かつ低価格になり，多くのプロセッサ，広大なキャッシュ空間，高速な内部スイッチ，および多数のディスクドライブが備えられた大規模ストレージ装置[1),2)]が登場している．また，ストレージネットワーク[3)]の普及により，大規模ストレージ装置は複数のサーバから共有される傾向にあり，**図 1** に示すようなストレージを中心としてその周辺にサーバが接続されるストレージ中心のシステム構成が広く採用されるようになりつつある．

豊富な計算機資源を有する大規模ストレージ装置を中心とするシステム構成においては，IO インテンシブなデータ処理を，サーバ上の計算機資源を用いて実行するのではなく，大規模ストレージ装置の有する計算機資源の一部を用いて実施することに，以下のような利点が見受けられる．

(1) **サーバ–ストレージ間 IO 帯域におけるボトルネックの解消：** 一般にストレージ装置は高い内部帯域を有するのに対し，サーバ–ストレージ間の IO 帯域（外部 IO 帯域）は比較的低い．サーバ上で IO インテンシブなデータ処理を行う場合，外部 IO 帯域が飽和し，ボトルネックとなりやすい．IO インテンシブなデータ処理をストレージ装置内で実行することにより，このようなボトルネックを解消することができる．

(2) **物理情報に基づく IO 最適化制御：** IO インテンシブな処理の高速化には物理情報を用いた IO 最適化制御が不可欠である．仮想化された記憶

† 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo

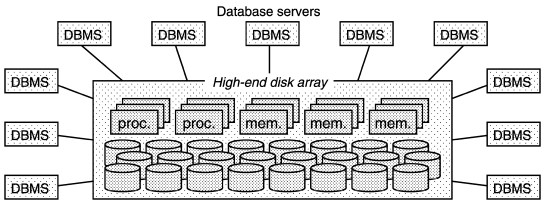


図 1 ストレージ中心のシステム構成
Fig. 1 A storage-centric IT system.

空間へアクセスするサーバ上のアプリケーションには，物理情報の取得が難しい．一方，ストレージ装置内では個々のドライブの物理情報を容易に取得することができるため，データ処理の IO をより最適化し，高いディスク並列度を得ることが可能である．

(3) **ストレージ側資源スケジューリング：** ストレージは複数のサーバから共有されるため，ストレージ装置の資源スケジューリングは単一のサーバに閉じず，システム全体の最適化は難しい．むしろ，ストレージ装置においてデータ処理を実行することにより，システム全体を見渡したグローバルなスケジューリングがより容易となる．

(4) **データとその処理機能の共有資源化：** データとその処理機能をストレージ装置に置き，共有資源と見なしシステム全体で共有することにより，個々のサーバにおけるデータ処理を軽減させ，システム設計を容易化し，ひいてはシステムの TCO 削減が期待される．

すでにバックアップ作成やスナップショット等の多くの機能[3)~6)]が，ストレージ装置の持つ豊富な計算機資源を有効に活用してストレージ装置内で実行されるようになっており，商用的にも広く受け入れられている．現状においては，これらのデータ管理機能は，かつてサーバ上で行われていた低レベルのデータ処理がストレージ上で実行されているにすぎないが，ストレージ装置の有する計算機資源はより拡大しており，今後ストレージ装置が，DBMS の一部機能等のより高度なデータ管理機能を取り込む可能性は十分にある．

以上の技術背景を鑑み，本論文では，オンラインデータベース再編成機能を有する高機能ディスクストレージである自己再編成ストレージ（SRS: Self-Reorganizing Storage）システム☆を提案し，その有効性を検証する．再編成はデータベース管理における重要な機能の 1 つであり，その処理はきわめて IO インテンシブである特色を有する．また，高可用性の要請から再編成はオンラインで実行可能であることが必須となりつつある．SRS におけるオンライン再編成方式自体は，すでに商用の DBMS 等で採用されている分離再編成[7)~11)]なる方式に基づいているが，本論文ではデータと再編成機能がストレージ内に共存することの利点を追求する．すなわち，ストレージ装置が有する豊富な IO 帯域と高い IO 処理能力を活用し，並列パイプライン化データ処理，物理アドレスレベル IO スケジューリングならびに独自の高速ログ適用処理なる技術を導入することにより，再編成を高速に実施することを目指す．著者の知る限り，データベース再編成をストレージ上で実行することを提案し，その有効性を検証する研究はこれまで行われていない．

本論文の構成は以下のとおりである．2 章ではストレージ技術ならびにデータベース再編成の観点から本論文の動機を論じる．3 章では SRS の基本設計を述べ，さらに，再配置とログ追い付きを高速化する詳細設計を解説する．4 章では商用 DBMS である HiRDB[12)] ならびにオープンソース DBMS である MySQL[13)] を対象とした SRS の試作機の実装を紹介する．5 章では試作機による TPC-H および TPC-C ベンチマークを用いた性能評価実験を紹介し，有効性を論じる．6 章では関連研究を紹介する．7 章では本論文をまとめる．

2. 動　　機

2.1 ストレージ技術

本節ではストレージにおける IO 帯域，プロセッサおよびソフトウェア開発コストの観点から本論文の提案の動機を述べる．

IO インテンシブなアプリケーションの性能は IO 帯域に強く依存し，通常データベースは複数のストライプ化されたディスクに格納されるため，ディスク並列度が性能向上の鍵となる[14),15)]．ストレージ装置の内部に目を向けると，現在主流の 3.5 インチディスクドライブの転送帯域は 90 年代より年率 40%で向上しており，現在では 0.1 GB/s に迫りつつある[16)]．また，ハイエンドのディスクストレージは高度な内部通信機構を有しており，その帯域は高く，**表 1** に示すとおり 64–68 GB/s の内部転送帯域を有する例もある．一方，ストレージ装置の外部に目を向けると，ストレージ–サーバ間の相互接続である SAN については，主流プロトコルであるファイバチャネルの帯域は 0.2 GB/s である．ファイバチャネルの速度向上は年率 25%程度であり，2005 年中の 0.4 GB/s 製品の市場投入に向けてプラグフェストが行われている[17)]．また，サーバ

☆ 本論文における自己再編成とはストレージ装置内で再編成が実行されることを意味する．再編成の自律的な判断に関しては，本論文の対象外とする．

表 1 大規模ストレージの最大構成仕様
Table 1 Maximum specifications of gigantic disk storage.

Model	EMC [1] DMX3000	Hitachi [2] TagmaStore
# of processors	116	128
Cache memory size	256 GB	128 GB
Internal data bandwidth	64 GB/s	68 GB/s
# of drives	576	1152
Physical capacity	84 TB	332 TB

の IO バス[18] に関しては，現在では最大 1.0 GB/s の PCI-X が主流である．近年，レーン数 * 0.25 GB/s の帯域を有する PCI Express が市場投入されたが，バスの速度進化は周辺機器のそれに比べて比較的緩やかであることから，依然として IO バスがボトルネックとなる状況は継続すると推測される．以上のことから，ストレージ内の IO 帯域はより高性能化しているのに対し，ストレージ外の IO 帯域は比較的貴重化しており，高いディスク並列度を得ようとしても，ストレージ外の IO 帯域がボトルネックとなる可能性が高いことが分かる．

高いディスク並列度を引き出すには，ディスクの物理情報を用いた IO 最適化制御が不可欠である[19)~21)]．現在のストレージ環境においては，ディスクドライブの記憶空間はそのままサーバ上のアプリケーションに提供されることはなく，ディスクアレイ，スイッチ，LVM 等における複雑な仮想化機構[22] を経て提供される．また，その記憶空間は通常，複数のサーバから共有される傾向にある．仮に膨大な投資によってストレージ外の IO 帯域を拡大したとしても，現在の仮想化，共有化が進展したストレージ環境においては，サーバ上のソフトウェアが十分にディスクの物理情報を獲得することは難しく，またディスクは共有資源であることから IO 制御は複数のサーバに分散化されるため，グローバルな IO 最適化制御は困難となる．以上のようなストレージを取り巻くハードウェア技術，システム技術の潮流を鑑みるに，ストレージ内で IO インテンシブなデータ処理を実行することは自然な流れであり，本論文の提案には合理性があるといえる．

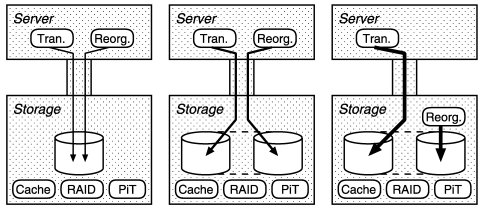
表 1 に示すように，今日ではハイエンドのディスクストレージは豊富なプロセッサ，広大なキャッシュ空間を有している．ストレージは通常これらの計算機資源を RAID 演算やキャッシュ制御に用いる．すべての計算機資源が常時利用されているわけではなく，未利用の資源の一部を活用することにより，サーバフリーバックアップ[3]，PiT（Point-in-Time）コピー[4]，メインフレーム・オープンシステム間のコード共有[5]，固定コンテンツ用アーカイブのためのブロック共有[6] 等のデータ処理がすでにストレージ内で実施されるようになっている．すなわち，ストレージ装置内にデータとその管理機能を同居させ，ストレージ装置が計算機資源を適宜管理機能に割り当てるという意味において，データ管理機能のプロビジョニングが実現されつつあるといえる．現状において，ストレージ上で実行される管理機能は単純で低レベルなものが主流であるが，デバイス技術の進展と，ストレージ装置の有する計算機資源が拡大していることから，近い将来，ストレージ装置上で DBMS のコードの一部が動作することは十分に可能性がある．本論文の提案は，未来指向の研究の第 1 歩として，その潜在的な有効性を検証するものである．

再編成コードは DBMS の実装に依存するため，ストレージ上の再編成ソフトウェアを DBMS およびそのバージョンごとに開発する必要がある．この点に関してはさらなる調査を必要とするが，現状においても以下のように本論文の提案を実現可能なものとする状況が見受けられる．第 1 に，DBMS の実装に依存する部分に関しては，再編成コードを DBMS ごとに個別に開発する必要がある．しかし，再編成全体を見渡した場合，主要な制御部分はほぼ共通であり，データアクセス部分に関してもある程度カプセル化することが期待できる．現に，4 章で述べるように，著者は HiRDB と MySQL という 2 つの異なる DBMS に対して SRS 試作機を実装したが，多くのコードが共通化もしくはカプセル化が可能であった．また，近年の仮想計算機技術の進展を見るに，ソフトウェアの実行環境はより抽象化される傾向にあり，LPAR[23] のような仮想化機能がストレージ装置に導入されると，再編成コードをストレージ装置において実行することはより容易となると考えられる．第 2 に，現在，いくつかの独立ベンダは再編成ユーティリティを複数の DBMS 向けに開発している[8),9)]．独立ベンダの再編成コードも DBMS に依存すると思われるが，商用製品を継続的に出荷していることから，DBMS へのコード依存性が必ずしもきわめて重大な経済的問題ではないと考えられる．

2.2 データベース再編成

本節ではデータベース再編成の観点から本論文の提案の動機を述べる．

記憶装置上のデータは，更新によってその構造が劣化する恐れがあり，構造劣化によりデータアクセスの性能は著しく悪化する可能性がある．たとえば，B 木においては，ディスクアクセスの観点から論理的に隣接したページが記憶装置上も物理的に隣接しているこ



(a) Concurrent Reorg. (Conventional) (b) Split Reorg. (Conventional) (c) Split Reorg. (SRS)

図 2 データベース再編成のアーキテクチャ
Fig. 2 Architectures of database reorganization.

とが望ましい．しかし，B 木の特定のページに偏ったレコード挿入を実施すると，当該ページは満杯となり分割されるため，物理的に離れた位置に新たなページを獲得し，レコードが格納される．このようなページ分割を繰り返すと，徐々にページの物理位置と論理的な鍵値との相関は低下することから，B+木の走査は記憶装置上ではランダムアクセスとなるため，検索性能が大幅に低下する恐れがある[24]．データベースにおいては，この問題は深刻であることから，ストレージ上のデータを再配置することにより劣化した構造を回復し性能を改善する再編成はデータベースに不可欠な機能である．

一般に，再編成はデータの再配置のために膨大な IO を要するため，負荷はきわめて大きく長時間の処理を必要とする．このためデータベースの構造劣化と再編成のコストを慎重に見極めて，再編成を実施するかどうかの判断をすることがデータベース管理者には求められる．構造劣化による性能悪化と再編成コストをモデル化することにより，長期的なデータベース運用において必要な再編成の間隔や時間を見積もり，最適化を試みる研究が行われてきた[25)~30)]．

近年ではデータベースにはきわめて高い可用性が要請されていることから，サービス継続中に再編成を実行可能なオンライン再編成[31)] に関する研究が行われてきた．これまでに提案されているオンライン再編成はサーバ上のソフトウェアによって実施され，その方式は主に同時実行再編成（Concurrrent Reorganization）と分離再編成（Split Reorganization）の 2 つに分類することができる．同時実行再編成[32)~40)] は，**図 2** (a) に示すように，排他制御により DBMS 上においてユーザのトランザクションと再編成を同時実行する．一方，分離再編成[7)~11)] は，図 2 (b) に図示するように，データベースの複製を作成し，元のデータベースではトランザクション処理を，複製データベースでは再編成を実施する．また，再編成終了後に，再編成中のトランザクションによる更新を複製データベースに反映する必要がある．いずれの方式においても，トランザクション処理性能への副作用を最小化することが求められると同時に，膨大なデータを効率良く高速に処理することが必要となる．しかし，2.1 節で述べたように，ストレージ外の IO 帯域は比較的貴重であり，トランザクション処理とデータベース再編成をサーバ上で同時に実行する場合，外部 IO 帯域がボトルネックとなりやすい．ゆえに，サーバ上のデータベース再編成については，IO 処理の競合は避けられないことから副作用を十分に防ぐことは困難であり，また同時にその高速化も容易ではないという問題が存在する．

本論文の提案は，図 2 (c) に示すとおり，ストレージ装置上でデータベース再編成を実行することにより，外部 IO 帯域における IO 競合を回避するとともに，物理情報を利用した IO 最適化制御を行い，高速な再編成を実現する．データとその再編成をストレージ内で共存させることにより，SRS は DBMS の記憶空間管理を軽減させるため，情報システムの複雑性の軽減に寄与すると思われる．

広く知られているように，多くの DBMS は，データベースの構造劣化を防ぎ再編成時期をなるべく遅らせるために，複雑な記憶管理機構を実装し，数多くの設定パラメータを設けている．このことは逆に DBMS の開発コストを増大させており，またデータベース管理を困難にしている．再編成をストレージ上で実行する本論文の提案は，逆に，DBMS にとってのデータベース再編成を軽減することから，DBMS の記憶管理を単純化し，設定パラメータを削減することにつながり，最終的には IT システム全体のコスト削減に資すると考えられる．

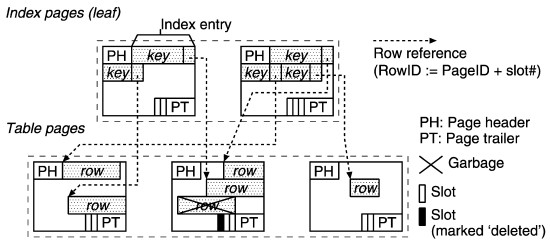
3. 自己再編成ストレージ

3.1 データ編成とログ取得

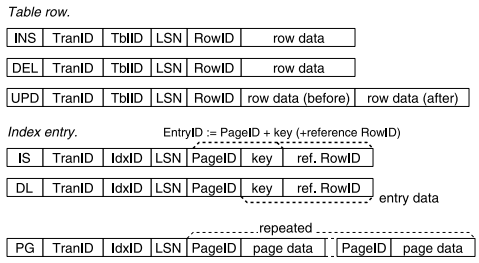
自己再編成ストレージの設計を述べる前に，データベースにおいて広く用いられているデータ編成，ならびにログ取得に関して簡潔に述べる．

3.1.1 データ編成

一般にデータベースにおいては，表および索引は表空間（tablespace）に格納される．表は行から構成される．一方，索引は索引エントリから構成され，多くの場合，B 木に基づく構造を有する．索引エントリは鍵，および対応する行への参照からなる．以降，本論文では単にエントリという場合，索引エントリを指すものとする．多くのデータベースの実装はクラスタリング機能を有している．クラスタ化された表，すなわちクラスタ化索引を有する表においては，行はクラス



(a) Page layout and reference



(b) Log record format

図 3 データ編成とログ取得
Fig. 3 Data organization and logging.

タ鍵順に配置されることが期待される．

表空間はストレージ上に配置された固定長ページの配列であり，各ページはページ ID なる識別子を有する．多くの実装ではページは最小の IO 単位である．**図 3** (a) にページの構成と参照の例を示す．ページ内はヘッダ，データ部，スロット部，トレイラから構成される．ヘッダとトレイラはページ ID，表 ID もしくは索引 ID，最新のページ更新の論理時刻，割当て済みの行やエントリ数等のページ管理情報を有する．データ部は行もしくはエントリのデータが，N-配列ストレージモデル（NSM: N-array Storage Model）[41] で格納される．スロット部の各スロットはデータ部内の行もしくはエントリへのページ内参照を有する．一般に，表ページでは，行が削除された場合，単にスロットに削除済みの印が付けられ，行が占めていた空間はすぐには再利用されない．表ページにおける行の挿入および削除は，他の行の移動をともなわずそのスロット番号が変化しないことから，行はページ ID とページ内のスロット番号の組合せで同定することが可能であり，この組合せ（ページ ID，スロット番号）を行 ID（RowID）と呼ぶ．一方，索引ページでは，ページ内ですべてのエントリはつねに鍵順に整列される．エントリが挿入もしくは削除された場合，ページ内の他のエントリが移動し，そのスロット番号が変化する．ゆえに，索引ページ内のスロット番号はエントリの同定には使用することができない．このため，唯一性制約のある索引のエントリの同定には（ページ ID，鍵）の組合せが用いられ，一方，それ以外の索引のエントリの同定には（ページ ID，鍵，対応する行への参照）の組合せが用いられる．これらのエントリ同定の組合せをエントリ ID（EntryID）と呼ぶ．

多くのデータベースの実装では，索引エントリにおける対応する行への参照には行 ID が用いられる．ゆえに，表空間内で行が移動して行 ID が変化すると，対応する索引エントリを更新して，当該エントリが正しく行を参照できるようにする必要がある．

また，エクステント（extent）なる表や索引の割当て単位を有する実装も存在する．エクステントは表空間内で隣接したページ群を意味する．同一エクステント内のページに格納された行もしくはエントリは，同じ表もしくは索引に属する．

表空間内の一部☆のページは，ディレクトリ（directory）なる表空間自身を管理するためのメタ情報を格納するために用いられ，ディレクトリページと呼ばれる．ディレクトリページが管理するメタ情報としては，空きページリスト，エクステント割当て表，空き・満杯ページのビットマップ等がある．

3.1.2 ログ取得

データベース内の表および索引へのすべての更新は，ログに記録される．ログはログレコードの配列であり，各レコードは更新の論理時刻であるログ順序番号（LSN: Log Sequence Number）を有する．多くの実装においては，ログレコードは 1 つの行もしくは索引エントリの更新を記録する．すなわち，行の更新と，対応する索引エントリの更新は，それぞれ別のログレコードに記録される．ログレコードは更新対象の行もしくはエントリを，行 ID もしくはエントリ ID をもって指定する．

図 3 (b) に本論文での議論の対象とするログレコードを示す．行の更新に関しては，3 種類のログレコードが存在し，`INS`，`DEL`，`UPD` はそれぞれ行の挿入，削除，更新を意味する．各ログレコードは，トランザクション ID，表 ID，LSN，行 ID および行データから構成される．`UPD` のみ更新前と更新後の 2 つの行データを有する．一方，索引エントリの更新に関しては，2 種類のログレコードが存在し，`IS`，`DL` はそれぞれエントリの挿入，削除を意味する．各ログレコードは，トランザクション ID，索引 ID，LSN，ページ ID およびエントリデータ（鍵，参照行 ID）から構成され

☆ たとえば HiRDB においては，ページ長を 16 KB，エクステント長を 512 KB とした場合，12 GB 以上の表空間におけるディレクトリページの占有率は，0.015%未満である．

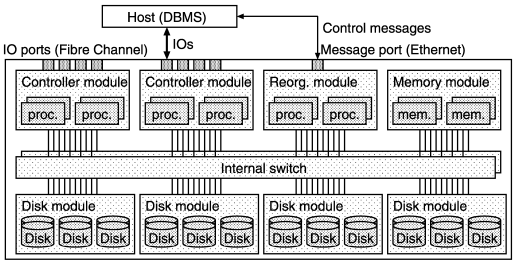


図 4 SRS のシステムアーキテクチャ
Fig. 4 SRS system architecture.

る．たとえば，SQL 等のデータ操作により表へ行が挿入される場合，ログにはまず行挿入の INS が記録され，続いて対応する索引へのエントリ挿入の IS が記録される．

多くのデータベースの実装においては索引は B 木に基づく構造を有しており，挿入によってページが 2 つに分割される，もしくは削除によって複数のページが 1 つに併合される等の場合がある．本論文では，このような振舞いを構造変化（structural change）と呼び，特殊なログレコードである PG によってページデータ単位で記録されるものとする．ログレコード PG は，トランザクション ID，索引 ID，LSN，および複数の更新ページ情報（ページ ID，ページデータ）から構成される．たとえば，ページ P1 が 2 つのページ P2，P3 に分割された場合，以下のログレコードが出力される．

{PG，トランザクション ID，索引 ID，LSN，
P1，P1 更新前データ，P2，P2 更新後データ，
P3，P3 更新後データ }

なお，本論文では簡単のため，構造変化は索引のみで発生し，表では発生しないものと仮定している．VSAM（Virtual Storage Access Method）[42] や索引構成表（IOT: Index-Organized Table）[43] 等，いくつかのデータベースの実装はより高度に構造化された表を有しており，当該表においては構造変化が発生する場合がある．本論文において 3.4 節で述べる索引の構造変化に関する議論は，このような高度に構造化された表にも直接応用することができるため，上記の簡略化は本提案の適用範囲を限定するものでないと考える．

3.2 システムアーキテクチャ

自己再編成ストレージ（SRS）の基本設計を述べる．**図 4** に SRS のシステム構成を示す．通常，高機能ディスクアレイは，高帯域内部スイッチ，ディスクモジュール，メモリモジュール，制御モジュールから構成される．制御モジュールは複数のプロセッサ，IO ポート，メッセージ用のポート等から構成され，RAID 等の仮想化制御，キャッシュ制御を司る．メモリモジュールは他のモジュールから共有され，主にキャッシュデータや仮想化の変換表の保持等に利用される．SRS ではデータベース再編成のためのソフトウェアを制御モジュール上で実行することが可能であり，特に，再編成ソフトウェアが動作する制御モジュールを再編成モジュール（Reorganization Module）と呼ぶ．SRS では，ディスクアレイ内の制御モジュールにおける RAID 等の機能によってサーバに提示される論理ディスクである論理ユニット（LU: Logical Unit）をデータベースの表空間として利用することを前提とする．

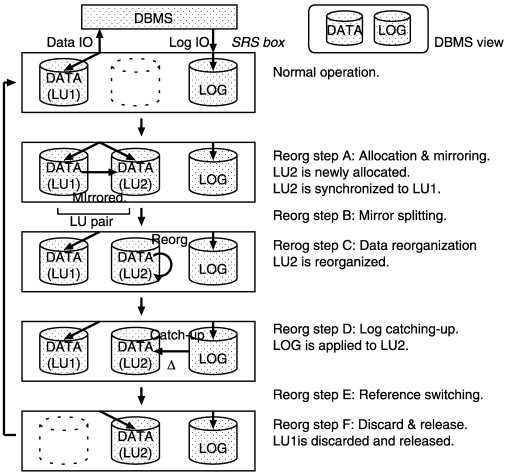


図 5 再編成の手続き
Fig. 5 Reorganization steps.

SRS においては，再編成命令を通信インタフェースを通じて再編成モジュールに与えることにより，分離再編成に基づくオンライン再編成を開始する．再編成命令は再編成の対象となる表空間と，再編成に関する設定パラメータ（目標充填率等）を指定する．再編成命令が再編成モジュールに到着すると，再編成モジュールは**図 5** に示す以下の手順により再編成を実施する．

A. 複製の作成 表空間として利用されている LU（以後 LU1 と呼ぶ）に対して，同じ構成の LU（以後 LU2 と呼ぶ）を動的に作成し，LU1 にミラーさせる．すなわち，LU1 上のデータは LU2 に複製され，LU2 は LU1 の同期複製となる．

B. ミラー対の分離 LU 間の同期が確立した後，SRS は再編成対象となる表空間を処理している DBMS に静止化（quiescence）を要求する．DBMS は静止化要求に従い，当該表空間に関連する新規トランザクションの受付を停止し，実行中のトランザクションが終了するのを待ち，バッファのフラッシュを行う．静止化が完了した後，SRS は RAID のマッピ

ングを書き換え当該表空間を構成する LU 対のミラーを切り離し，DBMS のアクセスが LU1 のみを参照するように変更し，再度 DBMS へ静止化の解除を要求する．DBMS は当該表空間の静止化を解消し，新規トランザクションの受付を開始する．これにより LU2 はデータベースレベルでの一貫性のある PiT コピーとなる．

C. データの再配置 SRS は LU2 に対して，データの物理配置を変更することにより，再編成を実施する．この際，表の行および索引のエントリの移動履歴を記述した行 ID 変換表（RCT: RowID Conversion Table）および索引エントリ ID 変換表（ECT: EntryID Conversion Table）を作成する．RCT は（旧行 ID→ 新行 ID）の列，ECT は（旧エントリ ID→ 新エントリ ID）の列である．この処理の間，DBMS は LU1 を参照してユーザのトランザクション処理を実行する．

D. ログ追い付き C. データの再配置の実行中に DBMS のトランザクションにより LU1 は更新されているため，SRS はその間に記録されたログを LU2 へ適用することにより，LU2 を LU1 へ論理的に追い付かせる．この際，ログは LU1 のアドレス空間に対して記述されているため，RCT および ECT を用いることにより，当該ログを LU2 に適用可能なように変換する．ログ追い付きの完了時点で，SRS の要求により DBMS は表空間を再び静止化する．

E. 参照の切替え DBMS は D. におけるログ追い付き後の静止化状態を引き継ぐ．SRS は RAID のマッピングを書き換え DBMS のアクセスが LU2 を参照するように切り替え，DBMS へ静止化の解除を要求する．DBMS は当該表空間の静止化を解消し，新規トランザクションの受付を開始する．

F. 複製の解放 SRS は LU1 のデータを破棄し，LU1 を解放する．

A.，B.，E.，F. は DBMS の有する静止化機能，およびストレージ装置が有する RAID やプロビジョニング等の仮想化機能，PiT コピー作成機能を用いて実現することができるため，本論文では詳細を述べない．一方，C. および D. の手続きは本論文が提案するストレージの新しい機能であり，実質的にこの手続きを高速化することが再編成全体の高速化につながるため，以降の節においてその詳細を述べる☆．

分離再編成は複製を利用することにより，トランザクション処理と再編成を分離し，排他やディスクアクセスの競合を避けることを目的とする．加えて，SRS においては，再編成処理をストレージ内で実行することにより，貴重な外部 IO 帯域のデータ転送を削減し，再編成によるトランザクション処理性能への副作用をなるべく回避することが期待される．E. においてログ領域への DBMS のログ書き込みと SRS のログ読み込みが競合するが，一般に，表空間のデータと異なりログ書き込みが性能に与える影響は小さく，両者の負荷ともシーケンシャルであることからディスクアレイのキャッシュによりログ書き込みへの副作用は無視できるものと想定する．

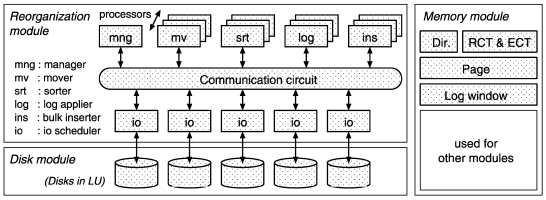


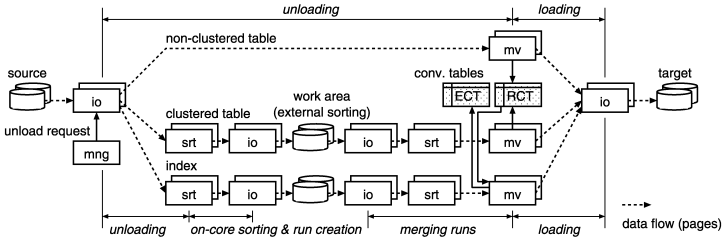
図 6 ソフトウェア構造
Fig. 6 Software structure.

図 6 に SRS においてデータの再配置およびログ追い付きを行うソフトウェア構成を示す．再編成モジュール内では移動プロセス，整列プロセス，ログ適用プロセス，挿入プロセス，IO プロセスおよび管理プロセスが配置される．管理プロセスはつねに 1 つ存在し，他のプロセスの管理を行う．IO プロセスはディスクごとに存在し，担当するディスクの IO スケジューリングを行う．その他のプロセスはプロセッサごとに起動可能であり，データ再配置およびログ追い付きに用いる．メモリモジュール上の共有メモリ空間にはページ，ディレクトリ情報，RCT，ECT，およびログウィンドウのバッファを設ける．

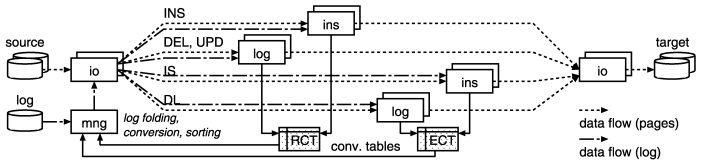
3.3 データ再配置の高速化

データの再配置はアンロード・ロード方式[44] を基礎とする．すなわち，データは表空間からアンロードされ，クラスタ化表や索引に関しては鍵により整列され，再び表空間にロードされる．再編成後には，表空間内で表，索引がそれぞれ連続した領域に目標充填率で割り当てられ，さらに，クラスタ化表および索引に関しては鍵の順序性が物理的に保たれる．再配置は原則として新領域再編成（new-place reorganization）[34] 方式により行い，アンロード対象の再編成元空間とロード対象の再編成先空間は別の領域とする．この際，

☆ 分離再編成においては，ログ追い付きに際して，DBMS のトランザクション処理に追い付くことができない恐れや，ログ追い付き中に再び構造劣化が発生する恐れがあるが，SRS においては，3.3 および 3.4 節に述べる方式により，データ再配置，ログ追い付きを高速化することにより，このような事態の可能性は大きく減少すると考えられる．



(a) Replacement



(b) Log catching-up

図 7 並列パイプライン化データ処理
Fig. 7 Parallel pipelined data processing.

SRS は表空間の同じ LU 内の空き領域を再編成先とする LU 内再編成（intra-LU reorganization）と，表空間の再編成元 LU とは別に同一構成の LU を作成しこれを再編成先 LU とする LU 間再編成（inter-LU reorganization）の 2 つの選択肢を有する．

索引は表に対して行 ID による参照依存性を有するため，再編成後の索引エントリが適切に再編成後の行を参照できるように再配置を実施する必要がある．SRS では再編成対象の表空間の組に対し，表の再配置と索引の再配置の 2 ステップに分けて処理を行う．データの再配置は以下の手順で行われる．

C.1 再配置計画 ディレクトリ情報，再編成命令の充填率目標を元に，アンロード後の必要空間量を見積もり，データの再配置計画を立て，空間を確保する．

C.2 表の再配置 すべての表ページの再配置を実施し，同時に行の移動を記述する RCT を構築する．

C.3 索引の再配置 すべての索引の葉ページの再配置を実施し，同時に RCT による索引エントリの行 ID 変換を実施し，さらにエントリの移動を記述する ECT を構築する．その後，再配置された葉ページを元に，枝ページおよび根ページを再構成する．

C.2 および C.3 が主要な手続きであり，当該手続きの高速化を図って，SRS はストレージ内部の豊富な IO 帯域と高い IO 処理能力を有効に利用することを目的として，並列パイプライン化データ処理と物理アドレスレベル IO スケジューリングを実施する．

3.3.1 並列パイプライン化データ処理

一般に，サーバ上の DBMS により行われるデータベース再編成については，問合せ負荷と競合することから，十分なデータ処理の高スループット化の試みがなされていない場合が多い．SRS はアンロード，整列，ロードの手続きをパイプライン化して処理することにより再編成の高速化を図る．非クラスタ化表に関しては，再編成元空間からのアンロードと再編成先空間へのロードがパイプライン化により同時実行され，1 回の走査で再編成が完了する．クラスタ化表および索引に関しては，再編成元空間からアンロード，主記憶上での高速整列☆および一時領域への整列ラン書き込みがパイプライン的に重畳化して実施され，その後，一時領域からの整列ランの読み込みと併合整列，再編成先空間へのロードが同様に重畳化して実施され，2 回の走査で再編成が完了する．

図 7 (a) に示すデータ処理方式をもって，データ再配置は実施される．すなわち，表ページおよび索引ページは，アンロードを実施する IO プロセスから，他のプロセスを経由して，ロードを実施する IO プロセスまでのデータ流として処理される．再編成に先立ち，管理プロセスはディレクトリ情報を参照し，アンロードにおける表および索引の走査において読み込むべきページ列を取得し，ページ列をページ ID 順に整列する．通常，ディレクトリ情報は表空間に対してきわめて小さいため，この処理にかかる時間も少ない．その後，アンロードのため，管理プロセスはページ列に基づき逐次的にページ IO 命令を各ディスクを担当する IO プロセスに送る．非クラスタ化表に関しては，IO プロセスがディスクからページを読み込み，移動プロ

☆ 整列ラン長は利用可能な主記憶上のバッファ長に依存する．

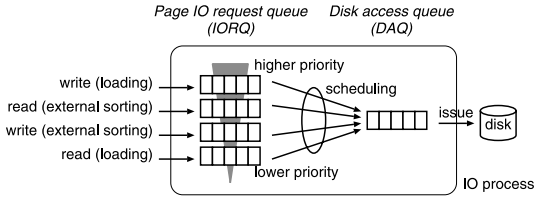


図 8 IO プロセスにおける物理アドレスレベル IO スケジューリング
Fig. 8 Physical IO scheduling at IO process.

セスが行の配置変更を実施するとともに RCT へ移動を記録し，最後に IO プロセスが再びディスクへページを書き込む．加えて，クラスタ化表に関しては，整列プロセスがクラスタ鍵による整列を実施する．索引に関しては，整列プロセスが索引鍵による整列を実施し，移動プロセスが RCT を参照して索引エントリの行 ID の変換を行い，索引エントリの移動を実施するとともに，ECT へ移動を記録する．

3.3.2 物理アドレスレベル IO スケジューリング

SRS では LU のメンバディスクごとに IO スケジューリングを実施することにより，IO 並列度を高め，再編成の高速化を図る．**図 8** に IO プロセスにおける物理アドレスレベル IO スケジューリングを図示する．各 IO プロセスは割り当てられたディスクへの IO のみを取り扱い，4 つのスケジューリング待ちのためのページ IO 要求キュー（IORQ）と 1 つのディスクアクセス待ちのディスクアクセスキュー（DAQ）を用いて IO スケジューリングを行う．ページ IO は読み込み（アンロード），書き込み（整列），読み込み（整列），書き込み（ロード）の順でより高位の優先度を有し，各優先度に IORQ が対応する．IORQ では以下のスケジューリングが実施される．第 1 に，シーケンシャリティを高めるため，ページ IO 要求は LBA 順に整列される．第 2 に，ディスクアクセスのオーバヘッドを削減するため，アクセス先の隣接したページ IO 要求は 1 つの IO 要求にまとめられる．IO プロセスは，DAQ 内に要求がある場合，これをディスクに発行する．DAQ が空になると，上述のスケジューリングが行われ，優先度に基づき DAQ は IORQ に接続され，IO 要求が渡される．

簡単のため，上記では LU はパリティなしのストライピング構成されていると仮定しているが，容易に RAID5 等の RAID 編成にも応用可能である．

以上の手続きにより，SRS は並列パイプライン化データ処理と物理アドレスレベル IO スケジューリングを行うことにより，ディスクアレイ内の内部 IO 帯域と IO 処理能力を有効に利用して，高速なデータベース再編成を図る．

3.4 ログ追い付きの高速化

データ再配置の終了後，再配置中に DBMS のトランザクションによる更新によって記録されたログを再編成後の LU2 に適用し，LU2 を論理的に LU1 に追い付かせる必要がある．この際，ログは LU1 に対して記録されており，そのログレコードは LU1 に依存した行 ID もしくはエントリ ID による参照を有することから，直接 LU2 に適用することはできない．SRS では RCT および ECT を用いてログを LU2 に適用可能なように変換する．具体的なログレコードに関して，必要な変換を以下に説明する．

まず，表の更新ログの変換に関して述べる．`DEL` および `UPD` は適用対象の行を行 ID によって指定することから，`DEL` および `UPD` を再編成後の LU2 に適用するためには，RCT によって再編成前の古い行 ID を新しい行 ID に変換する必要がある．一方，`INS` も同様に対象行を行 ID によって指定するが，`INS` の指定する行は再編成開始後に挿入されたものであるため，RCT には当該行 ID は含まれておらず，RCT による行 ID 変換は不可能である．ゆえに，`INS` に関しては直接ログ適用を行うのではなく，ログレコードから行データを取り出し，データベースの挿入操作を用いて行を挿入する．`INS` ならびに `DEL` の適用の後，挿入もしくは削除された行は RCT に反映される．

次に，索引の更新ログの変換に関して述べる．`DL` は適用対象エントリをエントリ ID，すなわち（ページ ID，鍵）もしくは（ページ ID，鍵，参照行 ID）をもって指定することから，`DL` を再編成後の LU2 に適用するためには，ECT によって再編成前の古いページ ID を新しいページ ID に変換するとともに，再編成前の古い参照行 ID を新しい参照行 ID に変換する必要がある場合がある．一方，`IS` も対象行をエントリ ID によって指定するが，`INS` と同様の理由により，ECT によるエントリ ID 変換は不可能であるため，直接ログ適用を行うのではなく，ログレコードからエントリデータを取り出し，データベースの挿入操作を用いてエントリを挿入する．ただし，挿入後のエントリが適切に対応する行を参照できるように，エントリ内の参照行 ID をエントリ挿入に先立ち，RCT によって変換する必要がある．`IS` ならびに `DL` の適用の後，挿入もしくは削除されたエントリは ECT に反映される．なお，ページ分割等の構造変化を意味するログレコード `PG` は，表空間内で該当するページのエントリが移動し，エントリのエントリ ID が変更することを意味する．ログ追い付きにおいて `PG` を検出した場合

には，PG におけるページデータを解釈し，その構造変化によるすべてのエントリの移動を ECT に反映する．これにより，後のログの適用において，ECT によるエントリ ID 変換が適切に行われる．

3.4.1 ログ畳み込みとログ整列

ログ追い付きを高速に実施するため，適用に先立ち，ログは主記憶上のログウィンドウバッファ内においてログ畳み込み（log folding）およびログ整列（log sorting）の 2 つの手続きを用いて処理される．ログ畳み込みは適用するべきログレコードの数を削減する手法である．表の更新ログに関しては，同一の行を更新するログレコードは 1 つ以下のログレコードに集約され，同様に，索引の更新ログに関しては，同一の行を参照するエントリを更新するログレコードは 2 つ以下のログレコードに集約される．ログ畳み込みが行われた後，RCT，ECT による行 ID およびエントリ ID 変換が行われる．変換の後，ログ整列が実施される．すなわち，ログ適用における IO の連続性を高めるため，ログ中のレコードは行 ID もしくはエントリ ID によって整列される．

上記の手続きについて，構造変化を考慮する必要がある．ログウィンドウバッファ中のログ内に，構造変化を表すレコード PG が含まれている場合，ウィンドウ中においてレコードの参照するエントリ ID が変化するため，ログ畳み込みを実施することができない．SRS では PG を含むトランザクション実行中のログレコードに関してのみ，ログ畳み込みおよびログ整列を実施せず，LSN 順に逐次適用を実施する．

行 ID およびエントリ ID 変換，ログ畳み込みおよびログ整列を統合したログ追い付きの手続きを以下に示す．

D.1 ログの読み込み ログから LSN 順に未適用のログレコードを，ログウィンドウバッファが満杯になるか，ログの終端に達するまで読み込む．その後，D.2 以降に移動し，ログの処理と適用を実施する．ログ読み込み中に PG を検出した場合，ログの読み込みを停止し，PG を含むトランザクションの直前のログレコードまでをもって，D.2 および D.3 のログ処理ならびにログ適用を実施する．引き続き，PG を含むトランザクション実行中のログレコードに関しては，LSN 順に逐次的に行 ID およびエントリ ID 変換を行い，表空間に適用する．その後，再びログの読み込みを開始する．

D.2 表ログの処理と適用 ログウィンドウバッファ内で，第 1 に，INS，DEL および UPD を抽出し，行 ID による整列を実施し，**図 9** に示す規則をもって畳み込みを行う．すなわち，同一行を更新するログレコードを 1 つ以下のログレコードに集約する．第 2 に，DEL および UPD に関しては，RCT をもって古い行 ID を新しい行 ID に変換する．その後，DEL および UPD を新しい行 ID によって整列し，表空間に適用する．DEL によって削除された行を RCT から削除する．第 3 に，INS に関しては，データベースの挿入操作を実施する．非クラスタ化表に関しては，行データを抽出し，一括して表へ挿入する．クラスタ化表に関しては，行データを抽出し，クラスタ鍵で整列し，一括して表へ挿入する．INS によって挿入された行を RCT に追加する．

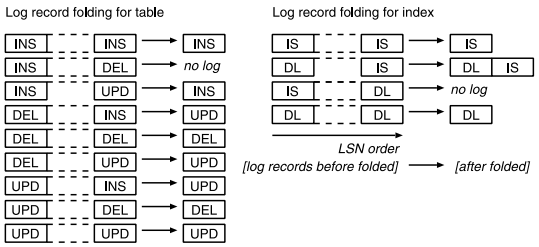


図 9 ログ畳み込み
Fig. 9 Log folding.

D.3 索引ログの処理と適用 ログウィンドウバッファ内で，第 1 に，IS および DL を抽出し，参照行 ID による整列を実施し，図 9 に示す規則をもって畳み込みを行う．すなわち，同一行に対応するエントリを更新するログレコードを 2 つ以下のログレコードに集約する．第 2 に，DL に関しては，ECT をもって古いエントリ ID を新しいエントリ ID に変換する．その後，DL を新しいエントリ ID によって整列し，表空間に適用する．DL によって削除されたエントリを ECT から削除する．第 3 に，IS に関しては，エントリデータを抽出し，鍵で整列し，一括して索引へ挿入する．IS によって挿入されたエントリを ECT に追加する．

D.4 終了判定 未適用ログ量が閾値以上の場合，D.1 に戻る．未適用のログが存在して，その量が閾値未満である場合，DBMS に静止化を要求し，D.1 に戻る．すべてのログが適用済みである場合，ログ追い付きを終了する．

上記の手続きにより，再編成中に記録されたログを再編成後の空間に安全に適用することができる．ログ畳み込みにより適用ログ数を削減することが可能であるとともに，ログ整列によりログ適用時の IO の連続性を高めることが可能となる．INS および IS の適用に関しては，データベースの挿入操作を用いた一括挿

入を実施するが，多くの場合，一括挿入は連続的な IO を発行するため，ログ適用の効率化に大きく寄与する．

3.4.2 並列パイプライン化データ処理と物理アドレスレベル IO スケジューリング

ログ追い付きの手続きのうち，主要な処理である D.2 および D.3 の高速化のため，図 7 (b) に示すデータ処理方式によりログ追い付きを実施する．表ページおよび索引ページがデータ流として処理される．同時に，データ流中の各ページには，当該ページに適用するべきログレコードが添付される．管理プロセスはログレコードを読み込み，ログウィンドウバッファ内で処理し，IO プロセスに送信する．IO プロセスはログレコードを解釈し，必要なページを読み込む．この際，ページと対応するログレコードはデータ流に統合される．`DEL`，`UPD` および `DL` に関しては，ログ適用プロセスがログレコードをページに適用し，RCT および ECT を更新し，IO プロセスが適用されたページを書き込む．一方，`INS` および `IS` に関しては，挿入プロセスがデータベースの挿入操作を用いて，行もしくはエントリを一括挿入する．万が一，一括挿入時にページ分割等の構造変化が発生した場合，即座に ECT へ反映する．

IO プロセスは 3.3.2 項で述べた方式により物理アドレスレベル IO スケジューリングを実施する．ログ追い付きについては，IORQ として高優先度の書き込みキューと低優先度の読み込みキューの 2 種類のみを取り扱う．物理アドレスレベル IO スケジューリングは，読み込み・書き込み競合を回避し，IO 処理のオーバヘッドを削減することにより，ログ追い付きにおいても有効に機能すると考えられる．

4. 試作機の実装

3 章で述べた設計に基づき，HiRDB Version 7 [12] および MySQL 4.1.1 [13] を用いて SRS の試作機を実装した．本論文では主に HiRDB を用いた試作機に焦点を当てて述べる．

HiRDB は銀行業務，証券取引，鉄道運営，コンテンツ管理等に用いられている商用 DBMS である．試作機の実装に際しては，非クラスタ化表とクラスタ化表を考慮した．非クラスタ化表では通常，行は挿入順に配置されるが，クラスタ化表では索引を用いてクラスタ鍵順に配置されることが期待される．また，索引は B+木により構成され，葉ページのエントリのみが行への参照を有する．ディレクトリは表および索引のエクステント割当てを管理するとともに，各索引の根ページのページ ID を管理する．

一方，MySQL は代表的なオープンソース DBMS である．試作機の実装に際しては，InnoDB データベースエンジンを用いた．InnoDB では通常，表および索引とも B 木に基づくデータ構造に格納される．

4.1 構造劣化の例

HiRDB において起こりうる構造劣化の例を以下に示す．

ケース A. 疎な表の全表走査 行の削除操作については，HiRDB は該当するページ内のスロットに削除フラグを立て，行データが占めていた空間を回収しない．これは，削除操作の応答性能を高めるとともに，将来ロールバックが発生した場合にも高速に行を回復させることを目的としている．ページやエクステント全体の行が削除されたとしても，ただちにはページやエクステントは開放されないため，全表検索を実施する場合，削除された空間も走査する必要が生じる．また，同じ表に対して，挿入と削除が繰り返される場合，削除された空間は再利用されず，結果として当該表に割り当てられた物理空間は疎となり，全表検索を実施する場合，実データ長に対して本来必要以上の空間を走査する必要が生じ，問合せ応答性能の悪化につながる．この場合，問合せ応答性能を回復させるために，再編成により削除によって生じた空間を回収し，充填率を高める必要がある．

ケース B. 高水準位を越える挿入 非クラスタ化表への挿入操作については，HiRDB は主記憶上に管理される高水準位（HWM: high-water mark）なるポインタを利用する．HiRDB はオンライントランザクション処理における挿入操作の応答性を重視しており，挿入操作は HWM を活用した楽観的なプロトコルを採用している．HWM は各表について割り当てられた最大のページ ID を指しており，挿入操作は HWM を参照することにより，空きページ探索を行うことなく，高速な挿入を実施することが可能となる．その反面，HWM が表空間の終端まで到達した場合，HWM は機能せず，挿入操作に対して表空間の先頭から空きページを検索する必要が生じ，挿入の応答時間が悪化する．この場合，挿入の応答性能を回復させるために，再編成により HWM を適切な位置に移動する必要がある．

4.2 試作機と実験システム

性能評価実験のために SRS 試作機を実装し，さらにデータベースサーバを接続した実験システムを構築した．実験システムの概要を**図 10** に示す．SRS 試作機は内部スイッチとしてのファイバチャネルスイッチ，ディスクモジュールとしての 4 台の JBOD（合計 24 台

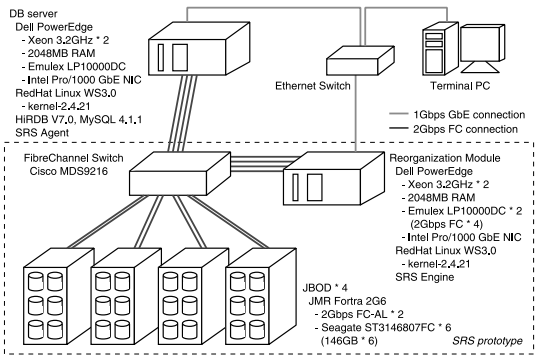


図 10 実験システム
Fig. 10 Experimental system.

のディスクドライブ☆），および再編成モジュールとしての PC サーバにより構成される．当該 PC サーバは 2 個の 3.2 GHz Intel Xeon プロセッサ，2048 MB の主記憶を有する．IO アクセス用に 4 本の 2 Gbps ファイバチャネルによりファイバチャネルスイッチと接続されるとともに，制御通信用にギガビットイーサネットワークにも接続される．さらに，Linux オペレーティングシステム上で，再編成モジュールの実装であるソフトウェア（SRS エンジン）が動作する．データベースサーバ上では HiRDB ならびに MySQL が動作するとともに，再編成モジュールとの連携を管理する専用のデーモンソフトウェア（SRS エージェント）が動作し，ギガビットイーサネットにより SRS エンジンと通信する．また，専用の PC 端末がデータベースサーバおよび再編成モジュールを管理する．管理端末はデータベースサーバ，再編成モジュールから情報を収集し，ユーザが設定した条件により，再編成モジュールに再編成命令を発行する機能を有する．

当該実験システムは再編成モジュールの有効性の検証に焦点を当てており，ディスクアレイ外部へ LU を提示する RAID 機能の模擬は SRS 内部では行わず，データベースサーバ側で Linux カーネルの `md` ドライバによるソフトウェア RAID 機能を利用し，LU を模擬する．再編成モジュールはデータベースサーバから `md` のマッピング情報を取得し，同じ LU を模擬する．

5. 性能評価

SRS の有効性を検証するために，データベースベンチマークである TPC-H および TPC-C をアプリケーションとして用い，性能評価実験を行う．本論文では主に HiRDB を用いた試作機の実験に焦点を当てて述べる☆☆．

5.1 構造劣化によるデータベース性能悪化と再編成によるデータベース性能回復

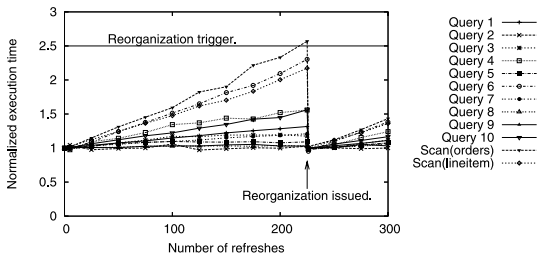
TPC-H および TPC-C を用いて，ストレージ上の構造劣化が DBMS の性能悪化に与える影響と，SRS の再編成による性能回復を計測する．この際，HiRDB の設定はすべてページ長を 16 KB，エクステント長を 32 ページ，共有バッファ長を 512 MB とし，ストライピングユニット長を 64 KB として 3 台のディスクからストライピング構成される LU を 2 つ作成し，それぞれ表，索引用の表空間として利用する．

TPC-H は典型的な意志決定支援アプリケーションのベンチマークであり，23 個の問合せと 2 個の更新操作が定義されている．RF1（New Sales Refresh Function）は新しい売上げ記録を `ORDERS` 表および `LINEITEM` 表に追加し，RF2（Old Sales Refresh Function）は古い売上げ記録を削除する．RF1 と RF2 の組合せによりデータウェアハウスの一部が更新されるが，データウェアハウス全体の大きさはほとんど変化しない．実験は以下のとおり行う．非クラスタ化表と索引をスキーマとして定義し，スケールファクタを 8 とした約 8 GB の初期データを生成する．RF1 および RF2 の組合せを 300 回実行することにより徐々にデータウェアハウスを更新し，25 回の更新ごとに Q.1 から Q.10 までの問合せと，`ORDERS` 表および `LINEITEM` 表の全表検索の応答時間の変化を測定する．RF1 および RF2 の更新率は 1%とし，DBMS のキャッシュ効果を除くため，問合せごとに共有バッファをクリアする．さらに，管理端末は問合せの応答時間を監視し，初期状態から 2.5 倍以上に悪化した場合，目標充填率を 100%とした再編成を SRS に命令する．測定の結果を**図 11** (a) に示す．

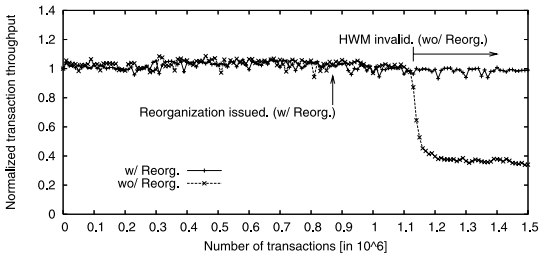
問合せのうち，Q.1，4，6，10 等の `ORDERS` 表もしくは `LINEITEM` 表に対する表走査を実施する処理の応答時間が更新回数に応じて，徐々に悪化することが分かる．これは，4.1 節で述べた構造劣化のケース A. に該当し，RF1 による挿入と RF2 による削除によって，空間の密度が低下し，表走査の効率が低下することによる．また，225 回目の更新の際，`ORDERS` 表の応答時間が 2.5 倍以上悪化したため，SRS により再編成が実施され，問合せの応答時間が改善する．以上のことから，SRS により構造劣化による段階的な性能悪化か

☆ 24 台のディスクドライブへのアクセスには十分なファイバチャネル帯域が確保されている．

☆☆ MySQL を用いた試作機の実装においても同様の結果を得ているが，その有効性に関しては紙面の都合から一部のみを紹介し，別稿に譲る．



(a) TPC-H



(b) TPC-C with migration task

図 11 データベース性能の悪化と回復
Fig. 11 Degradation and restoration of database performance.

ら性能を回復させることが可能であることが分かる．

一方，TPC-C はオンライントランザクション処理のベンチマークであり，5 種類のトランザクションが定義されている．純正 TPC-C ではつねに `Order`，`Order-Line`，`History` の各表の行数が単調に増加するが，実際のシステムにおける長期的な運用の下では，当該表のうち古いデータは定期的にデータウェアハウスやアーカイブ等に移送される．このような長期的なオンライントランザクション処理システムの振舞いを模擬するため，定期的に実行されるバッチとして移送処理を導入し，移送処理により `Order`，`Order-Line`，`History` の各表からトランザクションに影響を与えない古い履歴が削除されるようにする．長期的に `Order`，`Order-Line`，`History` の各表の行数をほぼ一定に保つように，移送処理によって削除される行数は適切に制御される．すなわち，移送処理が N トランザクションごとに実行される場合，当該移送処理は直前の N トランザクションによって追加された行数とほぼ同数の行を削除する．実験は以下のとおり行う．本実験のみ表空間長は 2 GB に限定し，非クラスタ化表と索引をスキーマとして定義し，ウェアハウス数を 16 とした約 1.5 GB の初期データを生成する．思考時間を 0 として，150 万トランザクションを実行し，1 万トランザクションごとのスループットを計測する．移送処理は 10 万トランザクションごとに起動される．実験は SRS による再編成を実施しない場合（**wo/Reorg.**）と，実施する場合（**w/Reorg.**）の測定を行った．**w/Reorg.** の場合，管理端末は HWM の位置を監視し，表空間の 95%以上に到達したことを契機として，目標充填率を 100%とした再編成を SRS に命令する．測定結果を図 11 (b) に示す．

実験開始後，いずれの場合もしばらくの期間はスループットが安定的な値を保っているが，**wo/Reorg.** の場合，113 万トランザクション近傍から急激に低下する．これは，4.1 節において述べた構造劣化のケース B. に該当し，113 万トランザクションまでは HWM ポインタを利用して挿入を実施しているが，それ以降は HWM ポインタが表空間の端に到達し，無効化されることによる．すなわち，表空間内には空き空間が存在するが，挿入は空き空間を探索する必要があるため，結果的にスループットが低下する．一方，**w/Reorg.** の場合，あらかじめ 88 万トランザクション近傍において，SRS により再編成が実施され，その後の HWM 無効化を防ぐことが可能となっており，スループットの低下を防ぐことができることが分かる．以上のことから，SRS により，構造劣化による急激な性能悪化を未然防止することが可能であることが分かる．

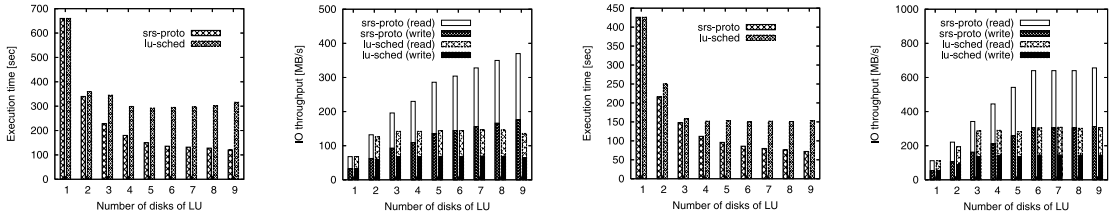
5.2 データ再配置の性能評価

SRS はディスクアレイの持つ内部 IO 帯域と IO 処理能力を効果的に活用して高スループットの再編成を実現するため，並列パイプライン化データ処理と物理アドレスレベル IO スケジューリングを行う．本方式の有効性を検証するため再配置の性能測定を実施する．なお，本実験では表データの再配置に焦点を当てる．

5.2.1 DBMS の再編成ユーティリティとの比較

5.1 節と同じ実験環境において，TPC-H および TPC-C のデータに対するデータ再配置の実行時間を測定する．TPC-H に関しては，スケールファクタを 8 とした約 8 GB の初期データに対して RF1 および RF2 による更新を 5 回行ったデータを対象とする．TPC-C に関しては，ウェアハウス数を 16 とした約 1.5 GB の初期データに対して 100 万トランザクションを実行したデータを対象とする．いずれも再編成の目標充填率を 100%とする．SRS の設定はページバッファ長を 512 MB とし，IO 制御のキュー長をディスクごとに 64 とし，LU は 3 台のディスクからストライピング構成されるものとする．この際，SRS 試作機による LU 間再編成（**srs-proto**）と，DBMS の再編成ユーティリティによる再編成（**dbms-utils**）の実行時間を比較する．**dbms-utils** の場合，同じディスク 3 台から構成される LU を一時領域として用いる．

図 12 に計測結果を示す．DBMS の再編成ユーティ



(a) Execution time (intra-LU) (b) IO throughput (intra-LU) (c) Execution time (inter-LU) (d) IO throughput (inter-LU)

図 13 ディスク並列度とデータ再配置性能の関係
Fig. 13 Disk parallelism effect on data reorganization performance.

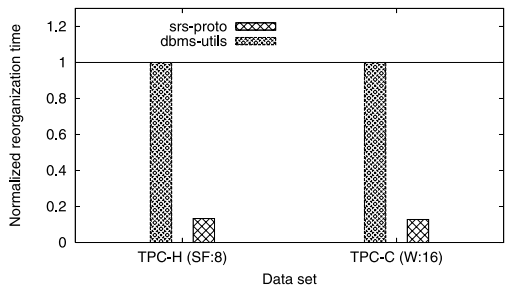


図 12 SRS と DBMS ユーティリティのデータ再配置の性能
Fig. 12 Data reorganization performance (SRS vs. DBMS utility).

リティと比較して約 13%の実行時間でデータ再配置を完了していることが分かる．このため，今日一般的に用いられている再編成ユーティリティと比較して，SRS の再編成により大幅な性能改善が可能であることが分かる．

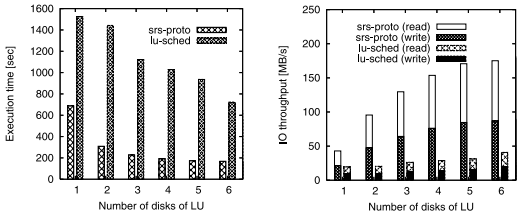
5.2.2 物理アドレスレベル IO スケジューリングの効果

ディスクアレイ内の IO 資源が有効に利用されていることを確認するために，前項と同じ実験設定において，TPC-H におけるスケールファクタを 16 とした初期データに対して，RF1 および RF2 による更新を 10 回行ったデータを対象に，目標充填率を 100%とした再配置を実施し，LU を構成するディスク台数を 1 から 9 まで変化させ，再配置の実行時間，IO スループットを測定する．SRS 試作機の再編成を測定するとともに，比較対象として物理アドレスレベル IO スケジューリングを行わない再編成（**lu-sched**）を測定する．この場合もデータ処理はパイプライン化して行われる．IO 制御はカーネル標準の `md` デバイスを用いて LU 単位で行い，`md` デバイスを有効に機能させるため，主記憶の一部をディスクキャッシュとして利用可能とする☆．

LU 内でデータ再配置を実施した場合の計測結果を**図 13** (a)，(b) に示す．ディスクの台数が少ない場合，両方の場合で同様の測定結果が得られており，方式間に大きな相違はないといえる．ディスクの台数が 3 台以上の場合，IO スループットの増加が **lu-sched** の場合では約 150 MB/s 程度で飽和しており，これ以上の実行時間の改善が見込めない．一方，**srs-proto** の場合では，ディスク 9 台においても実行時間，IO スループットが改善されており，最大で約 370 MB/s の IO スループットを達成している．**lu-sched** も LU 単位で IO スケジューリングを実施しており，**srs-proto** に比べ膨大なキャッシュ空間が利用可能であるが，オーバヘッドが無視できず，スループットの飽和に至っている．一方，LU 間でデータ再配置を実施した場合の計測結果を図 13 (c)，(d) に示す．LU 内再配置と同様の傾向が得られており，**srs-proto** の場合，最大で 660 MB/s という高い IO スループットが得られている．いずれの場合においても，ディスク台数が多い場合，**srs-proto** は **lu-sched** の約 2 倍のスループットを達成している．**srs-proto** ではディスク単位で IO 制御を行い，オーバヘッドの削減を図ることから，より高い IO 並列度を得ることができることが確認できる．本実験結果から，データ再配置においてディスクストレージの IO 制御能力および高い IO 帯域を利用するためには物理アドレスレベル IO 制御がきわめて有効であることが分かる．

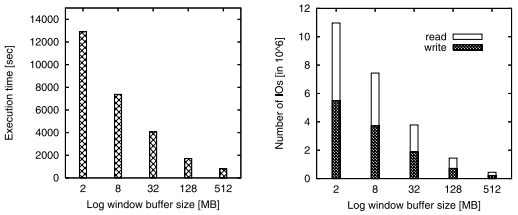
なお，別の実験環境において MySQL を対象とした実装を用いた場合の，LU 間のデータ再配置の実行時間および IO スループットの計測結果を**図 14** に示す．**lu-sched** に対して **srs-proto** の実行時間は大幅に改善されており，また高い IO スループットが達成されている．本論文の提案する物理アドレスレベル IO スケジューリングが，HiRDB だけでなく MySQL においてもきわめて有効に機能することが分かる．MySQL

☆ 通常，未使用の主記憶をカーネルのディスクキャッシュとして用い，LRU とエレベーションアルゴリズムに基づく制御を行う．本実験においては，およそ 1 GB 程度の主記憶がディスクキャッシュとして利用可能であった．



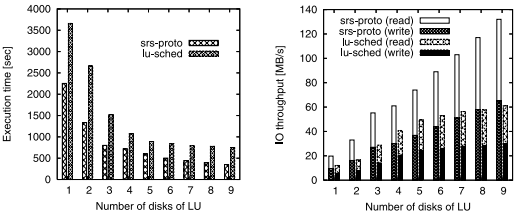
(a) Execution time (b) IO throughput

図 14 データ再配置性能（MySQL を対象とした実装）
Fig. 14 Data reorganization performance (Implementation using MySQL).



(a) Execution time (b) IO cost

図 15 ウィンドウ長とログ追い付き性能の関係
Fig. 15 Window size effect on catching-up performance.



(a) Execution time (b) IO throughput

図 16 ディスク並列度とログ追い付き性能と関係
Fig. 16 Disk parallelism effect on catching-up performance.

においては表データは B 木に基づくデータ構造に格納され，HiRDB のそれとは大きく構造が異なるが，提案手法が双方に有効に機能することから，その有効性が示される．

5.3 ログ追い付きの性能評価

SRS はログをウィンドウバッファ上でログ畳み込み，およびログ整列を用いて処理することにより，高速なログ追い付きを図る．本方式の有効性を検証するため，ログの追い付きの性能測定を実施する．なお，本実験では表データのログ追い付きに焦点を当てる．

本実験ではあらかじめ一定量のログを生成し，この適用を測定する．5.1 節と同じ実験環境において，TPC-C におけるウェアハウス数を 16 とした約 1.5 GB の初期データに対して 100 万トランザクションを実行する．その後，表空間のデータを同じ構成の LU からなる表空間に複製する．元の表空間においてさらに 100 万トランザクションを実行し，ログを取得する．その後，複製された表空間においてデータ再配置を実行し，RCT および ECT を生成し，引き続いてログを適用する．この際の SRS の設定はページバッファ長を 512 MB とし，IO 制御のキュー長を 64，LU は 3 台のディスクのストライピング構成とする．

ログ適用処理におけるウィンドウ長を 2 MB から 512 MB まで変化させ，ログ適用の実行時間を測定する．計測結果を**図 15** (a) に示す．この結果，ウィンドウ長の増大によって，大幅に実行時間を改善することが可能であることが分かる．

さらに，ログウィンドウバッファの効果を分析するため，各ウィンドウ長におけるログ適用に必要な論理 IO 数を測定する．図 15 (b) に結果を示す．ウィンドウ長の増大により，畳み込み処理によって適用ログの数自体が減少すると同時に，ログ整列により IO の連続性が高まり，同一ページへの更新が集約され，IO 数を大幅に低下させることが分かる．以上の実験結果から，ログ畳み込み，ログ整列を用いたログ追い付きが有効に機能し，実行時間を大幅に削減することが可能となることが分かる．

最後に，ウィンドウ長を 512 MB とし，LU を構成するディスク台数を 1 から 9 まで変化させた場合に，実行時間および IO スループットを計測した．比較対象として，5.2.2 項と同じく **srs-proto** と **lu-sched** 双方の IO スケジューリングによるログ追い付きを測定する．計測結果を**図 16** に示す．すべての場合で **srs-proto** が **lu-sched** より優位な結果が得られている．特に，ディスクの台数が 5 台以上の場合，**lu-sched** の場合では IO スループットの増加および実行時間の改善が鈍化している．一方，**srs-proto** の場合では，ディスク台数 9 台においても IO スループットが増加し，実行時間の改善に寄与していることが分かる．これは，**lu-sched** では LU 単位で IO 制御を実施しているため，ディスク台数の増加に応じた処理の並列度の向上が不十分であるのに対し，**srs-proto** ではディスク単位で IO 制御を行い，オーバヘッドの削減を図ることから，より高い並列度を得ることができるためと考えられる．以上の実験結果から，ログ追い付きにおいても本論文の提案する物理アドレスレベル IO スケジューリングが有効であると考えられる．

6. 関連研究

6.1 ストレージ技術

DBMS のコードの一部をストレージのプロセッサ上で実行する発想は，70 年代から 80 年代のデータベースマシンに関する研究に遡ることができる[45)~48)]．データベースマシンは高価な専用ハードウェアを必要としたが，見合った性能が得られなかったことから，最終的に産業界はデータベースマシンの研究から撤退するに至った．90 年代後半にはアクティブディスクもしくはインテリジェントディスクが提案された[19)~21)]．その発想自体はデータベースマシンと類似であるが，プロセッサとメモリがより高性能化し，より安価となった技術背景に支持された．以上の研究はいずれもアドホックな問合せやデータマイニング等をストレージ上のプロセッサで実行することを主眼としていたが，データベースの中心的なコードは一般に複雑で膨大であり，現時点においてもなお，そのようなコードをストレージに移譲することは現実的ではない．また，サーバとストレージの役割分担，すなわち，何をサーバが行い，何をストレージが行うべきかという問題に関して，これまで十分な議論はなされなかった．

一方，本論文はストレージ内においてデータベース再編成を実行することを提案する．問合せ処理やトランザクション処理等のデータベースの中心的なコードと比べ，再編成等のデータベース管理機能は DBMS から切り出しやすく，また比較的複雑ではないため，ストレージのプロセッサ上で実行することが可能である．トランザクション処理に代表される高度な処理を高価なサーバのプロセッサで実行し，再編成に代表される IO インテンシブで比較的単純な処理をストレージのプロセッサで実行するという意味において，本論文の提案は技術潮流から見ても自然な役割分担を提案するものである．

6.2 データベース再編成

同時実行再編成は 80 年代以降のオンライン再編成に関する学術研究の主要な課題であった[32)~40)]．同時実行方式は記憶空間の利用効率が高いという利点があるが，複雑な同時実行制御を必要とする．90 年代以降においては，ディスクドライブの記憶密度が急速に高まった背景から，産業界は，安価となった記憶容量を活用して複雑な同時実行制御を避け，分離方式によるオンライン再編成を開発してきた[7)~11)]．著者の知る限り，文献 11) は分離再編成に関する数少ない論文である．当該論文はファジー[49)] 再編成なる手法により，分離再編成における同期点を避けることを主眼としている．すなわち，複製の作成と，再編成をサーバにおいて同時実行することができる．

一方，本論文の提案手法はより単純な方式を採用しており，複製の作成，再編成とログ追い付き，参照切替えは静止化を用いて逐次的に行われる．本論文は，ストレージにデータベース再編成機能を移譲することの利点を明らかにすることに焦点を当てており，再編成を議論しているものの，過去の再編成に関する研究とは異なった目的を有する．

7. まとめ

本論文はデータベース再編成機能を有する高機能ディスクアレイである自己再編成ストレージ（SRS）を提案した．SRS はデータベース再編成をその内部で実施することにより，データベースの構造の効率性を保つことが可能である．ディスクストレージ内の高い IO 処理能力と豊富な IO 帯域を有効に利用するため，並列パイプライン化データ処理，物理アドレスレベル IO 制御，高速ログ適用処理を実施する．商用 DBMS ならびにオープンソース DBMS を対象とした試作機を実装し，性能評価実験を行った．実験の結果，提案手法によりデータベース再編成におけるデータ再配置，ログ追い付き双方において大幅な性能の改善が可能であることが示された．

自己再編成ストレージは，構造劣化の管理をサーバからストレージに移譲しようとするアプローチの提案である．より詳細なストレージとサーバの新しい役割分担の有効性について，検討を進めていきたい．

謝辞 本研究の一部は，文部科学省リーディングプロジェクト e-Society 基盤ソフトウェアの総合開発「先進的なストレージ技術」の助成により行われた．協力企業である株式会社日立製作所より多くの有益なコメントを頂戴した．感謝する次第である．

参考文献

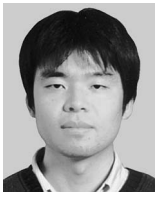
1) EMC Corp.: EMC Symmetrix DMX Series, Data Sheet (2004).
2) Takahashi, N. and Yoshida, H.: Hitachi TagmaStore Universal Storage Platform: Virtualization without Limits, White Paper, Hitachi Data Systems (2004).
3) 喜連川優：ストレージネットワーキング，オーム社 (2002).
4) Azagury, A., Factor, M., Satran, J. and Micka, W.: Point-in-time Copy: Yesterday, Today and Tomorrow, *Proc. 10th NASA MSST/19th IEEE MSST 2002*, pp.259–270 (2002).
5) EMC Corp.: EMC Mainframe Solutions

Guide, Engineering White Paper (2002).
6) Carleton, J.: Electronic Data Archiving and Retrieval in the Public Sector, Analyst Report, Frost & Sullivan (2004).
7) Oracle Corp.: Oracle Database 10g Online Data Reorganization & Redefinition, White Paper (2004).
8) Mohamed, A., Candia, G. and Sherwin, D.: Comparing Architectures of Online Reorganization, White Paper, Quest Software (2002).
9) BMC Software, Inc.: Easing the Pain of an IMS Reorganization, White Paper (2002).
10) Hitachi Ltd.: Nonstop operation: HiRDB. http://www.hitachi.co.jp/Prod/comp/soft1/global/prod/hirdb/nonstop.html
11) Sockut, G.H., Beavin, T.A. and Chang, C-C.: A Method for On-Line Reorganization of a Database, *IBM Syst. J.*, Vol.36, No.3, pp.411–436 (1997).
12) Hitachi Ltd.: Hitachi Relational Database Management System Solutions for Disaster Recovery to Support Business Continuity, Review Special Issue, Hitachi Technology (2004).
13) MySQL AB.: MySQL: The World's Most Popular Open Source Database. http://www.mysql.com/
14) Livny, M.: Multi-disk management algorithms, *Proc. SIGMETRICS 1987*, pp.69–77 (1987).
15) Patterson, D., Gibson, G. and Katz, R.: A Case for Redundant Arrays of Inexpensive Disks, *Proc.SIGMOD 1988*, pp.109–116 (1988).
16) HGST San Jose Research Center: HDD Technology Overview Charts (2003).
17) Fibre Channel Industry Association: FCIA Roadmap. http://www.fibrechannel.org/
18) Bhatt, A.: Creating a Third Generation I/O Interconnect, Intel Developer Network for PCI Express Architecture.
19) Riedel, E., Gibson, G. and Faloutsos, C.: Active Storage For Large-Scale Data Mining and Multimedia, *Proc. 24th VLDB 1998*, pp.62–73 (1998).
20) Acharya, A., Uysal, M. and Saltz, J.: Active disks: programming model, algorithm and evaluation, *Proc. 8th ASPLOS 1998*, pp.81–91 (1998).
21) Keeton, K., Patteson, D. and Hellerstein, J.: A case for intelligent disks (IDISKs), *SIGMOD Record*, Vol.27, No.3, pp.42–52 (1998).
22) Bunn, F. and Paglar, R.: *Storage Virtualization I, What, Why, Where and How?* SNIA Education (2004).
23) Barrick, D., et al.: Logical partitions on the IBM PowerPC, A Guide to Working with LPAR on Power5 i5 Severs, IBM Redbook (2005).
24) Watanabe, S. and Miura, T.: Reordering B-tree Files, *Proc. ACM SAC 2002*, pp.681–686 (2002).
25) Shneiderman, B: Optimum Data Base Reorganization Points, *Comm. ACM*, Vol.16, No.6, pp.362–365 (1973).
26) Yao, S.B., Das, K.S. and Teorey, T.J.: A Dynamic Database Reorganization Algorithm, *ACM Trans. Database Syst.*, Vol.1, No.2, pp.159–174 (1976).
27) Maruyama, K. and Smith, S.E.: Optimal Reorganization of Distributed Space Disk Files, *Comm. ACM*, Vol.19, No.11, pp.634–642 (1976).
28) Lohman, G.M. and Muckstadt, J.A.: Optimal Policy for Batch Operations: Backup, Checkpointing, Reorganization and Updating, *Proc. SIGMOD 1977*, p.157 (1977).
29) Batory, D.S.: Optimal File Designs and Reorganization Points, *ACM Trans. Database Syst.*, Vol.7, No.1, pp.60–81 (1982).
30) Fung, K.T.: A Reorganization Model Based on the Database Entropy Concept, *Comput. J.*, Vol.27, No.1, pp.67–71 (1984).
31) Lomet, D. (Ed.): Special Issue on Online Reorganization, *IEEE Data Eng. Bull.*, Vol.19, No.2 (1996).
32) Salzberg, B. and Dimock, A.: Principles of Transaction-Based On-Line Reorganization, *Proc. 18th VLDB 1992*, pp.511–520 (1992).
33) Zou, C. and Salzberg, B.: On-line Reorganization of Sparsely-populated B+-trees, *Proc.SIGMOD 1996*, pp.115–124 (1996).
34) Zou, C. and Salzberg, B.: Safely and Efficiently Updating References During On-line Reorganization, *Proc. 24th VLDB 1998*, pp.512–522 (1998).
35) Omiecinski, E. and Scheuermann, P.: A Global Approach to Record Clustering and File Reorganization, *Proc. 7th SIGIR 1984*, pp.201–219 (1984).
36) Omiecinski, E.: Incremental File Reorganization Schemes, *Proc. 11th VLDB 1985*, pp.346–357 (1985).
37) Omiecinski, E., Lee, L. and Scheuermann, P.: Performance Analysis of a Concurrent File Reorganization Algorithm for Record Clustering, *IEEE Trans. Knowl. Data Eng.*, Vol.6, No.2, pp.248–257 (1994).
38) Zabback, P., Onyksel, I., Scheuermann, P. and Weikum, G.: Database Reorganization in Par-

allel Disk Arrays with I/O Service Stealing, *IEEE Trans. Knowl. Data Eng.*, Vol.10, No.5, pp.855–858 (1998).
39) Thereska, E., Schindler, J., Bucky, J. and Salmon, B.: A framework for building unobtrusive disk maintenance applications, *Proc. 3rd USENIX FAST 2004*, pp.213–226 (2004).
40) Ghandeharizadeh, S. and Kim, D.: On-line Reorganization of Data in Scalable Continuous Media Servers, *Proc. 7th DEXA 1996*, pp.751–768 (1996).
41) Ramakrishnan, R. and Gehrke, J.: *Database Management Systems*, WCB/McGraw-Hill (2000).
42) Lovelace, D., Ayyar, R., Sala, A. and Sokal, V.: *VSAM Demystified*, IBM Redbooks (2003).
43) Oracle Corp.: Oracle 9i Index-Organized Tables Technical Whitepaper, White Paper (2001).
44) Sockut, G.H. and Goldberg, R.P.: Database Reorganization—Principles and Practice, *ACM Comput. Surv.*, Vol.11, No.4, pp.371–395 (1979).
45) Slotnick, D.: Logic per Track Devices, *Advances in Computers*, Vol.10, pp.291–296 (1970).
46) Su, S. and Lipovski, G.: CASSM: A cellular system for very large database, *Proc. 1st VLDB 1975*, pp.456–472 (1975).
47) Ozkarahan, E., Schuster, S. and Smith, K.: RAP—Associatice Processor for Database Management, *Proc. AFIPS 1975*, pp.379–387 (1975).
48) DeWitt, D. and Hawthorn, P.: A Performance Evaluation of Database Machine Architectures, *Proc. 7th VLDB 1981*, pp.199–214 (1981).
49) Moham, C. and Narang, I.: An Efficient and Flexible Method for Archiving a Data Base, *Proc. SIGMOD 1993*, pp.139–146 (1993).

(平成 16 年 12 月 20 日受付)
(平成 17 年 4 月 5 日採録)

(担当編集委員 宮崎 純)



合田 和生(正会員)

昭和 52 年生. 平成 12 年東京大学工学部電気工学科卒業. 平成 17 年同大学大学院情報理工学系研究科電子情報学専攻博士課程単位取得満期退学. 現在, 同大学生産技術研究所産学官連携研究員. 並列データベースシステム, ストレージシステムに関する研究に従事.

喜連川 優(フェロー)

昭和 30 年生. 昭和 53 年東京大学工学部電子工学科卒業. 昭和 58 年同大学大学院工学系研究科情報工学専攻博士課程修了. 工学博士. 同年同大学生産技術研究所第 3 部講師. 現在, 同教授. 平成 15 年より同所戦略情報融合国際研究センター長. データベース工学, 並列処理, Web マイニングに関する研究に従事. 平成 9 年, 10 年電子情報通信学会データ工学研究専門委員会委員長, 平成 11〜14 年 ACM SIGMOD Japan Chapter Chair, 平成 14 年, 15 年本会理事, 平成 15 年本会フェロー. 日本データベース学会理事, SNIA-J 顧問. VLDB Trustee, IEEE ICDE, PAKDD, WAIM 等ステアリングコミティメンバ.