DEWS2004 1-A-02

ウェアラブルコンピューティングのための追記型ファイルシステムの実装

小野田英樹† 波多野賢治† 宮崎 純† 植村 俊亮†

† 奈良先端科学技術大学院大学 情報科学研究科 〒 630–0192 奈良県生駒市高山町 8916–5
E-mail: †{hideki-o,hatano,miyazaki,uemura}@is.aist-nara.ac.jp

あらまし ウェアラブルコンピューティング環境では，小型コンピュータ上で能動型データベースを利用することにより，コンピュータが個人アシスタントのように動作することが望ましい．我々は，これまで能動型 DBMS を小型ディスクドライブ上に構築し，ポータビリティを持たせることを検討してきた．小型ディスクドライブは大容量化しており，個人情報の持ち歩き，データの蓄積は容易となってきている．しかし，デスクトップ用ディスクに比べ，シーク時間と回転速度が著しく劣るためデータアクセス速度に問題がある．本稿では，小型ディスク上で高速なデータアクセスを実現する手法として，ログを利用した追記型シーケンシャルアクセスを利用する．本手法を実装し，従来のページ単位アクセスによるファイルシステムとのデータアクセス性能に関する比較を行うことで，提案手法の有効性を示す．また，ログ追記型シーケンシャルファイルシステムを構築することによって得られる利点について考察する．
キーワード ウェアラブルコンピュータ，能動型データベース, 小型ディスクドライブ, 追記型ログファイルシステム, テンポラルクエリー

An Implementation of an Append-only File System for Wearable Computers

Hideki ONODA†, Kenji HATANO†, Jun MIYAZAKI†, and Shunsuke UEMURA†

† Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916–5, Ikoma, Takayama, 630-0192 Japan
E-mail: †{hideki-o,hatano,miyazaki,uemura}@is.aist-nara.ac.jp

Abstract Users desire their electronic assistance on wearable computers. In order to recommend information to users automatically, use of an active database is effective. Since an active database requires a large number of disk accesses, the system has to be able to access data at high speed from/to a tiny disk drive. However the data access speed of tiny disks is low because their seek time and rotational latency are inferior to those of desktop models. We make use of a log-structured file system for the sequential access that decreases the latency. In order to evaluate its performance and show the efficiency of the proposed log-based data access, we compare the proposed method with a conventional page-based one.
Key words Wearable computer, Active database, Tiny disk drive, Log file system, Temporal query

1. はじめに

近年，小型のノートパソコンや PDA(Personal Digital Assistant)，携帯電話（以下，これらをモバイル機器と呼ぶ）などの登場により，日常的にモバイル機器を携帯することが可能となってきている．モバイル機器は小型化，高機能化が進んでおり，将来的には GPS(Global Positioning System) やカメラを内蔵したウェアラブルコンピュータとして利用されると予想される．モバイル機器が，利用者のアシスタントとして動作する事への要求が高まりつつあるため，利用者が求めている情報を，状況に応じてモバイル機器が判断し，能動的に情報を提示する必要がある．同時に利用者の私生活をさまざま側面から記録したライフログを作成し，それを利用することで正確な情報の提供を行うことが可能であると考えられている．また，モバイル機器が利用者をアシストする場面は，日常生活の場から災害救助現場など幅広く求められており，ネットワークが無いような環境においてモバイル機器単体で動作する必要性がでてきている．機器単体で動作するためには，必要なデータをすべて蓄えることが可能な大容量なストレージが必要となる．大容量データをモバイル機器内に蓄えるストレージとして，低価格の小型ディスクドライブを利用することは適当である．

利用者が必要としている情報を能動的に提示するには，あらかじめ登録したルールに基づき動作し，ルールの追加，削除に

より，提示する情報のカスタマイズが可能な能動型データベースの利用が有効である．しかし，能動型データベースは通常のデータベースに比べ，イベント発生時のイベント処理，コンディションの評価およびアクションが，システム側で自動的に実行されるため，データの参照，イベントの監視など，ディスクアクセスが頻繁に発生し処理コストが高くなる．また，小型ディスクドライブはサーバ用ハードディスクドライブと比べシーク時間，回転速度が劣るため，データアクセス速度が低下する．そのため，迅速な情報の提供には小型ディスクドライブの特性に見合ったデータベースシステムの設計が必要となる．また，多様化するウェアラブル機器に対応する必要もある．ウェアラブル機器で起こりうる障害の一つに，メモリ内容の消失があげられる．メモリは一時的な記録でしかないため，電源障害が発生すると内容を消失してしまう．また，プロセスの異常終了時も同様にメモリ内容が消失する．

本稿では，これらの問題に対処するため，小型ディスク内のディスク制御プロセッサとバッファメモリを利用し，データベースシステムを小型ディスク内で動作させる．これにより，接続先モバイル機器への共通インターフェースを提供することが可能となる．また，ディスクアクセススピードの問題に対しては，ログファイルシステム [8] を利用することで，解決を行う．

提案するシステムでは，すべてのデータベース操作によって作成されたデータをディスク上に連続的に記録しているため，ディスクを最初から読み直すことでデータベースをリカバリすることができる．また，個人で利用するシステムでは，時間をパラメータとする問合せ要求が考えられる．提案するシステムでは，全データをディスク上に時系列で保存しているため，過去の情報を振り返って検索するテンポラルクエリを効率的に処理することが可能である．

提案する追記型ファイルシステムを実装するため，データ格納方法，クエリ処理方法，インデクスの作成方法について具体的に示す．

2. データベースと小型ディスクドライブ

2.1 データベース

データベースを利用する際，特に重要な項目が耐障害性である．データベースの障害によりデータの消失が発生すると，データを有効に利用できないばかりか，正しくない情報を提示してしまう可能性がある．したがって，データを消失させない仕組みの実装が重要となる．通常，データベースはログを利用してその問題を解決している．ログとは, データベース操作によりデータベースに変更が生じた際，記録する変更データのことである．

データを消失させないため，データベース管理システムは，作成したログを永続メモリに記録したのち持続メモリの更新を行う．永続メモリ上のログは条件が満たされるとディスクにフラッシュされる．システム障害が発生した場合，永続メモリ上にデータが保存されているためデータベースを復元することが可能となる．

2.2 小型ディスクドライブ

小型ディスクドライブ，例えば，日立のマイクロドライブは，構造的には従来のハードディスクドライブと同様ではあるが，回転速度などが十分であるとはいえない [1] [2] [3]．現状のディスクドライブの性能を表 1 に示す．

表 1　ディスクドライブの性能 [日立 HP より抜粋]

	マイクロドライブ	マイクロドライブ	サーバモデル
容量 (GB)	1	4	73.9
密度 (Gb/inch)	15.3	56.5	31.2
転送レート (Mb/s)	36 – 60	57.1 – 97.9	684 – 969
連続読み書き速度 (MB/s)	2.5 – 4.1	4.3 – 7.2	—
平均シークタイム (ms)	12	12	3.9
平均待ち時間 (ms)	8.33	8.33	1.99
回転速度 (rpm)	3600	3600	15037
データバッファサイズ (KB)	128	128	8192
メディア転送速度 (Mb/s)	38.8 – 59.9	57.1 – 97.9	—
インターフェース転送速度 (Mb/s)	11.1	33	—
消費電力 (mW)	420	360	12000

一般に，小型ディスクドライブは消費電力や耐衝撃性などの問題から高速なディスク回転速度が得られないため，ディスクアクセスが低速である．回転速度が上げられないことにより，ディスクヘッドが目的のレコード位置に移動するアクセス時間は約 20.33ms（平均シークタイム (12ms)+平均待ち時間 (8.33ms)）とシステム側からみると多大な時間を要する．

一方，ライフログを，MPEG4 [4] の規格である 384kbps のビットレートで記録した場合，カタログ上の性能ではディスクへの転送速度に問題はなく，24 時間記録した場合, 約 4GB の容量で収めることができる．

3. 提 案 手 法

半導体の集積技術の発展により，現在 1mm 四方のチップ上に 1 千万個以上のトランジスタを集積することができ，小型ディスクドライブ上に高速な CPU，大容量のメモリを搭載する事は技術的に問題がなくなっている [12]．したがって，図 1 に示すように，小型ディスク内のディスク制御プロセッサをデータベース管理システムの処理を行うプロセッサとして利用し，小型ディスクドライブ内のバッファを汎用メモリとして利用することが十分可能となっている．これによりデータベース処理を小型ディスクドライブ内で閉じて処理を実行することが可能になるので，モバイル機器に依存しないソフトウエアレベルの共通のインタフェースを提供することで，データベースシステムごとの移動を実現できる．

3.1 システムアーキテクチャ

3.1.1 記 録 方 式

我々は，小型ディスクドライブで，アクセス時間を短縮するために，追記型ファイルシステムを利用する．提案システムでは，データを追加する場合，ディスクに連続して追記していく手法をとる．このため，ディスク上にデータを隙間なく記録することで効率よくディスクを利用することができ，データ追加時には，従来のデータベースマネージメントシステムで起こるディスクのランダムアクセスは発生しない．そのため，アクセス時間内 (20.33ms) に約 115KB のデータを転送することが可能で，効率の良いデータアクセスが実現できる．データベースは，2.1 節で述べたように，ログが保存されていればリカバリが可能である．提案するファイルシステムでは，データ更新す

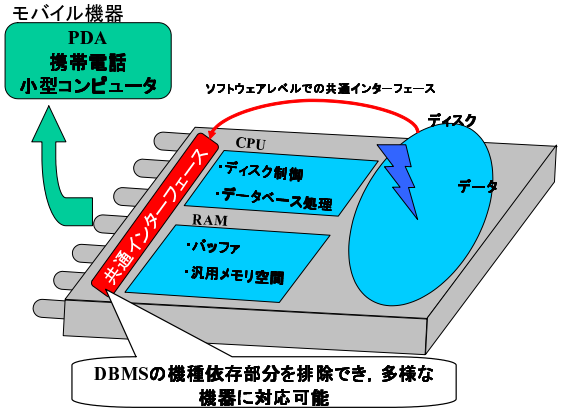

図 1　小型ディスク上のシステム構成

る場合，更新前データはディスクに残したまま，更新データをディスクの追記部分に記録する．また，データを削除する場合，実際にはデータをディスク上から消さない手法を採用する．したがって，データを追記していく方式はリカバリの観点から有効である．

また，サイズの小さいデータを頻繁にディスクに書き込む場合，ディスクアクセスのオーバヘッドが高くなるため，シーケンシャルアクセスの利点が十分に得られない可能性がある．このため，まとまったデータをまとめて書き込むグループコミット [7] の手法を採用し，効率的なデータ処理を実現する．図 2 に示すように，メモリ上にログバッファを設け，データの変更情報はログバッファに書き込んでいく．ログバッファがデータで満たされると，ディスクにデータを書き込む．ウェアラブル機器で扱うデータには，位置情報など，たとえデータが消失しても前後のレコードから推測が可能なデータがある．これらのデータは重要度が低いため，データの消失が発生しても問題がない．一方，所持金情報など，重要度の高いデータをメモリ上に蓄えると，障害によりデータが消失する危険性があるので，ディスクに強制的にフラッシュする．このようにデータにプライオリティ（重要度）を持たせることで，ディスクフラッシュの制御を行う．

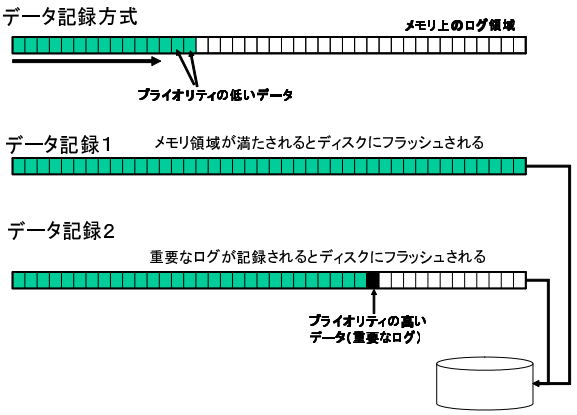

図 2　グループコミット

3.1.2　バッファメモリの利用

小型ディスクドライブ上のバッファを汎用メモリとして利用する．メモリ上にはディスクに格納したデータのデータ格納位置を示すインデクスを展開する．一般にインデクスはディスク

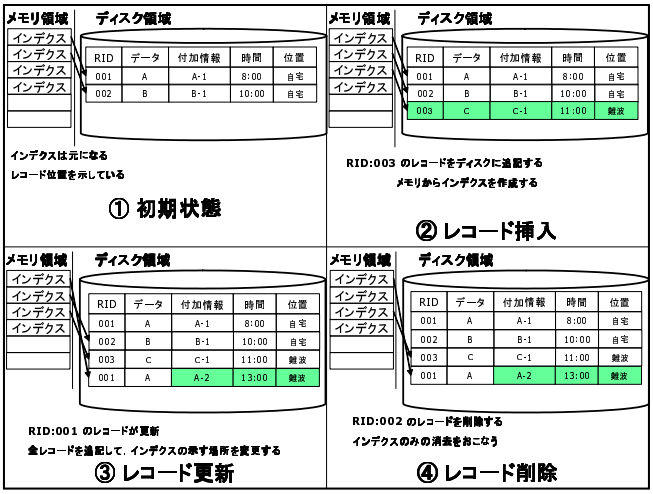

図 3　複製方式のデータ格納方式とインデクス

上のデータを検索する際，高速に処理を行うために利用する．しかし，インデクスは頻繁に更新されるため，ディスク上にインデクスを作成するとインデクスの読み込み，更新，変更の際，ディスクアクセスが必要となり，ランダムアクセスが発生する．このため，シーケンシャルアクセスの利点を十分に利用することができない．提案するファイルシステムでは，インデクスは小型ドライブ上のメモリに展開することで，この問題に対処する．

実際にウェアラブル機器に提案したディスクを装着して利用した場合を想定する．現在，GPS により計測できる精度は水平方向で 10cm になっている [5]．利用者が外出先で機器を利用した場合を想定すると，10cm の精度は必要ではなく，10m 程度で問題ないと考えられる．したがって，歩行時で約 10 秒に一度のデータの追加が行われることになる．また，外出時に利用するデータテーブルは，例えば，位置情報テーブル，店舗情報テーブル，利用者の動作を記録するプロファイルテーブルなどが挙げられる．上記 3 テーブルに 10 秒おきに操作がされると仮定すると，約 26,000 レコードが 24 時間で蓄えられ，このレコードに対するインデクスサイズは 105KB となり，小型ディスク内のメモリに収めることが可能である．自宅から持ち出す初期レコードは，利用者の個人データであると考えられるため，全レコード数が著しく増加することは考えにくく，全インデクスをメモリ上に構築することは可能である．

データ格納方式は 2 種類考えられ，インデクスの作成方法がそれぞれ異なる．処理形態をそれぞれを図 3，4 に示す．インデクスはディスクに格納されたレコード位置をキーとしている．

図 3 は，データ更新時，レコード全体を複製し追記する方式で，インデクスのポインタは追記されたレコードを示す．これを複製方式と呼ぶ．一方，図 4 は，データ更新時に更新された属性情報のみを追記する方式で，もともとのレコードへのポインタは保持したまま，更新された属性へのポインタを新たに作成し元レコードへのポインタにリストとして追加する．元データへのポインタと変更データの格納先へのポインタによりレコードの作成が可能である．これを差分方式と呼ぶ．

3.2　インデクス

通常インデクスはディスク上に記録されているため，インデクスを作成すると検索は速くなるが，データの更新処理，追加

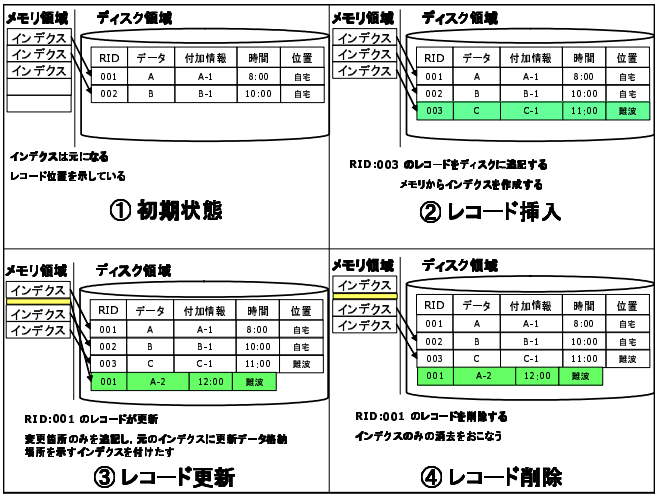

図 4　差分方式のデータ格納方式とインデクス

処理についてはインデクスの変更が発生するため処理が遅くなる．3.1.2 節で述べたとおり，個人で利用するシステムを想定しているので，全体レコード数は少ない．このため，メモリ上にインデクスを展開することができ，検索性能を高めることが可能である．しかし，メモリ上のデータは，電源障害時や，プロセスの異常終了時に消失する可能性がある．提案するシステムでは，定期的にインデクスを小型ディスク上に記録する．システム障害時，インデクスをディスクから読み込み，読み込んだインデクスのタイムスタンプを調べ，最新のインデクスが記録された後のログデータをディスク上から読み込むことで，インデクスを復元する．これにより，耐障害性について解決することができる．

3.2.1　複製方式

データの変更が発生した場合，変更の対象となるレコード全体を複製し，変更データを反映させた後，ディスクに追記する方式である．

（1）　挿入

データベースマネージメントシステムは，新たなレコードを挿入する時，新たに一意のレコード ID を発行する．レコード ID と，レコード長，新規レコードをひとつの連続データとしてディスクに追記する．データ格納後，格納位置へメモリからインデクスを張る．このとき，追記したレコードの先頭アドレスをインデクスとして記録する．従来のデータベースマネージメントシステムでは，データを挿入する際，格納可能な空き領域を検索するためランダムアクセスが発生し，格納に時間を要する．提案手法では，データを格納する追記場所があらかじめ決まっているため，ランダムアクセスは発生しない．そのため，効率よくデータを追加することが可能である．

（2）　削除

追記的にレコードを記述しているため，削除するレコード ID のみをディスクに追記する．その後，メモリ上のインデクスを削除する．提案ファイルシステムでは，実際にデータをディスク上から削除することはなく，削除情報のみをディスクに追記する．データ問合せ時には，インデクスを利用してレコード位置を求めるため，インデクス情報を削除することでシステム側からはデータが削除されたように見える．従来のデータベースマネージメントシステムは削除の実行時，ディスク上からデータの削除を行うが，提案システムでは削除処理も追記データして扱う．

（3）　更新

メモリ上のインデクスを利用して対象レコードを特定し，レコード ID を取得する．取得したレコード ID，更新情報を含むよう複製したレコードをディスクに追記する．更新前データをディスク上に残したまま，更新前データを示しているインデクスを追記されたレコード位置を示すように変更する．

（4）　問合せ

データ複製方式ではインデクス情報に最新のレコードの格納場所が書き込まれているため，ディスク上のレコード記録場所を前もって特定することができる．ディスクアクセスを行う前に，インデクスを用いて必要なレコードの格納場所を収集する．収集した位置情報をレコードの格納アドレス順にソートを行う．ソートを行うことで連続的なアクセスが可能となる．シーケンシャルアクセスでは，アクセス時間内 (20.33ms) に約 115KB のデータを転送することが可能であるため，問合せに不要な領域（必要レコードから次の必要レコードまでの領域）が，115KB 以下であれば，シーケンシャルアクセスによる読み飛ばしを行い，領域が 115KB 以上であれば，ヘッドを目的レコードへ移動させるランダムアクセスを行う．

複製方式では，処理が単純なうえ，インデクスはレコード数以上に増えないためメモリの節約が行える．しかし，レコード全体を複製して記録するため，ディスク使用効率は悪い．

3.2.2　差分方式

データ更新時には差分だけ追記する方式で，もともとのレコードへのポインタは保持したまま，更新された属性をディスクに書き込み，書き込まれた属性へのポインタを作成する．作成されたポインタはレコード全体を示すポインタに付け加える．

挿入，削除処理は，複製方式と同様の処理を行う．

（1）　更新

データ更新時にメモリ上のインデクスから，実レコードを呼び出し，レコード ID を取得する．レコード ID，更新する属性値をディスクに追記する．レコードに張られているインデクスに追記された属性へのポインタを付け加える．これによりディスクには更新データのみが追記された状態になる．差分方式は複製方式と比べると使用するディスク容量が小さくなる．

（2）　問合せ

メモリ上のインデクスから必要となるデータ格納場所をあらかじめ特定することが可能となる．さらに，インデクスに付け加えられたリストをたどることで，変更された属性の格納場所もあらかじめ特定することができる．これらの格納位置情報を格納順にソートを行った後，ディスクアクセスを行う．ディスクアクセスを行う際，問合せに不要な領域が 115KB 以下であれば，シーケンシャルアクセスによる読み飛ばしを行い，領域が 115KB 以上であれば，ヘッドを目的レコードへ移動させるランダムアクセスを行う．

差分方式では，ディスクに記録するデータは更新された属性データのみなので，ディスクの節約が可能である．しかし，属性の変更のたびにインデクスが作成されるため，メモリ使用において効率が悪い．メモリはディスクの比べ，記録単位あたりのコストが高いため，差分方式は複製方式で小型ディスク内に

収められないデータ量を取り扱う時に利用する．

4. 基本実験と追記型ファイルシステムの検証

3 節で提案したデータベースシステムを実装するにあたり，小型ディスクドライブのアクセス性能の検証を行った．また，インデクスを利用したデータ複製方式の追記型ファイルシステムを構築し，検索処理を行うことで，アクセス性能の検証を行った．

4.1 実　　装

現在のところ，内部リソースがユーザに公開された小型ディスクドライブは存在しないため，Linux 上でシステム構築，およびテスト行った．なお，Linux として，gentoo 1.4.1 (kernel Version 2.6)，PC として IBM ThinkPad (CPU:pentium3, Memory:384KB)，小型ディスクドライブとして，表 1 に示すマイクロドライブ (1 GB) を使用した．提案するデータベース管理システムの処理は PC 内部の CPU，メモリを使用して処理を行い，データは提案手法に基づいて小型ディスクドライブに格納した．

提案する追記型ファイルシステムを構築するため，C 言語の低水準入出力を利用して raw device としてディスクにアクセスを行う．低水準入出力ではディスクをシーケンシャルにアクセスすることが可能であり，OS があらかじめ備えるバッファリング機能を利用せず，直接ディスクとデータ転送を行うことが可能となる．一方，高水準入出力を利用すると，OS のバッファリング機能を利用し，ディスクの読み込み場所からバッファサイズに余裕がある限りデータを蓄積を行う．これによりバッファ内のレコードを利用する際，ディスクアクセスが必要なく高速に処理を行うことができる利点があるが，バッファに蓄えるためのデータを読み込むため，ディスクヘッドがバッファ容量分だけ移動する．問合せに不要な領域が 115KB を境に，シーケンシャルアクセスとランダムアクセスを使い分けるが，ヘッドがバッファ容量分移動しているため，最適なディスアクセスを行うことができない．

提案手法では raw device としてディスクを扱っているため，ディスクヘッドは最終アクセスを行ったディスクトラック上にあり，レコード追記場所へのアクセスを平均回転待ち時間 (8.33ms) で行うことが可能となる．

4.2 使用データ

実験では，ウイスコンシンベンチマークで利用されるレコードセットを用いてディスクアクセス性能について検証する．ウイスコンシンベンチマークは，208 バイトのデータを 1 レコードとしてデータの読み書きを行うテスト方式である．1 レコードは，整数型変数（4 バイト）13 個と 52 文字からなる文字型配列（52 バイト） 3 個から構成されている．レコードサイズが 208 バイトの場合，日立のマイクロドライブ（表 1）のカタログ上から計算を行うと，アクセス時間 (20.33ms) 内にシーケンシャルアクセスでは約 450〜700 レコードの読み込みあるいは書き込み処理を実行することが可能である．

4.3 基 本 実 験

提案する追記型データベースシステムの基本処理性能を検証するため，下記の三つの実験を行う．

4.3.1 実 験 内 容

（1） **ディスクの基本性能**

使用する小型ディスクの基本性能を調べるため，ディスク全体にデータを書き込む実験を行う．この実験結果から，以降の実験のパラメータを決定する．

（2） **レコード単位処理におけるランダムアクセスとシーケンシャルアクセスの性能比較**

ランダムアクセスとシーケンシャルアクセスで書き込んだ場合の処理時間について比較を行う．

図 5 に示すように，ランダムアクセスの場合，ポインタ情報から次レコード位置を特定し，ヘッドを移動させ次レコードの読み込みを行う．すなわち，レコードを読み込むたびにヘッドの移動が起こる．ランダムアクセスでは，レコード ID，データ，次レコードへのポインタを 1 レコードして取り扱う．また，シーケンシャルアクセスの場合，データは連続的に書かれているため，ヘッドの移動が起こらず，連続的にデータを読み込んでいく．シーケンシャルアクセスでは，レコード ID，データを 1 レコードとして取り扱う．

この実験では，ランダムアクセスのデータアクセス速度が，シーケンシャルアクセスに比べ著しく劣ることが予想できるため，両者の実測した処理速度の差分をとることで，アクセス時間を計測することができる．データベースの更新と，追加が頻繁に行われることにより，ランダムアクセスが必要となるデータが作成される．

（3） **ページ単位処理におけるページサイズ変更と応答時間**

ページ単位とレコード単位でのディスクアクセスの処理時間の比較を行う．レコードとは任意のサイズの単一データを示し，ページとは特定サイズのデータ集合で，一般的に数レコードのデータが含まれる．

頻繁にディスクアクセスを繰り返すと，ディスクアクセスのオーバヘッドが生じる．このため，ページ単位アクセスを利用し，一括してデータを書き込むことで，ディスクアクセス頻度を減らすことができ，効率よくデータ処理を行うことが可能となる．なお，ページ単位のサイズは，9 レコードを一括して書き込む 2KB のページサイズ，19 レコードを一括して書き込む 4KB のページサイズ 39 レコードを一括して書き込む 8KB のページサイズでアクセス速度を比較する．

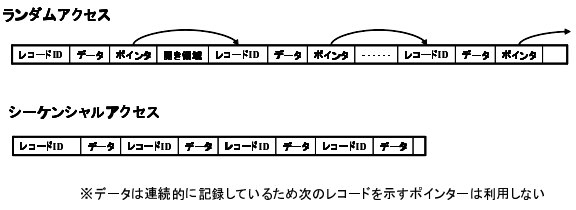

図 5　レコード単位のデータ格納模式図

4.3.2 検 証 結 果

はじめに，(1) のディスクの基本性能の検証を行った．10MB 単位のレコードを読み書きすることで計測を行った．結果を図 6 に示す．書き込みと読み込み双方でほぼ同様な結果が得られた．データ処理初期段階では小型ディスク内のバッファが利用されるため，高速に処理が行われている．また，ディスクサイズが 1GB のため，108 レコード (10MB×108 ≃ 1GB) で処理が終了している．ディスク内バッファを利用しない状態で，平均して約 6 秒で 10MB のデータを読み書きしているた

め，約 1.7MB/sec の読み書き速度となる．表 1 で示されたカタログ上の連続読み書き速度 (2.5 – 4.1MB/sec) と比較して，40%〜70% の性能が出ていることが判明した．1.7MB/sec での記録が行えるため，映像などの蓄積を行うモバイル機器で，MPEG4 [4] の規格である 384kbps のビットレートで記録を行うことが可能である．また，映像を記録させながら，問合せを行うことも可能であると考えられる．

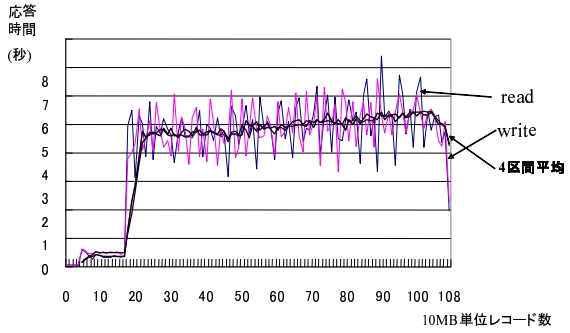

図 6　10MB 単位のデータでのディスク読み書き時間

次に，(2) のレコード単位処理におけるランダムアクセスとシーケンシャルアクセスの性能比較を行った．208 バイトのレコードセットを用いてシーケンシャルアクセスとランダムアクセスについてアクセス時間を図 7 に示す．シーケンシャルアクセスはランダムアクセスに比べ非常に応答時間が早い．シーケンシャルアクセスを 10,000 レコード分行った応答時間を図 8 に示す．1 秒間でランダムアクセスでは約 55 レコードの処理をしており，シーケンシャルアクセスでは約 9,000 レコードの処理を行っている．この結果から，アクセス時間がディスクアクセスに比べ多大な時間を要していることがわかる．

次に，計算によって求めた理論値と実測値の比較を行う．ランダムアクセスで 100 レコードを書き込んだ場合，実測値で 1,998.4ms を要している．一方シーケンシャルアクセスにおいて 100 レコードを書き込むのに要する時間は，実測値で 14.41ms である．これらの差分時間を書き込みレコード数 100 で割った値，19.84ms がアクセス時間となる．理論値は 20.33ms であるため妥当な数値と言える．

最後に，(3) のページ単位処理におけるデータサイズ変更による応答時間の計測を行った．4KB をページサイズとし，バッファを用いたディスクアクセスと，バッファを利用しないディスクアクセスの比較結果を図 9 に示す．4,000 レコードを書き込む際，ページ単位アクセスでは 68.9ms，レコード単位アクセスでは 227.7ms の時間を要している．バッファを用いる方がディスクアクセスのオーバヘッドを大幅に抑えることが可能となる．

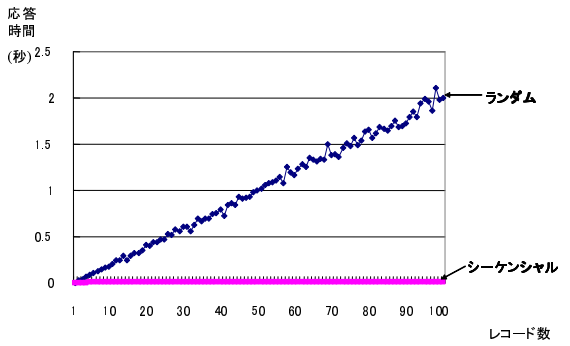

図 7　1 レコード処理によるランダムアクセスとシーケンシャルアクセス

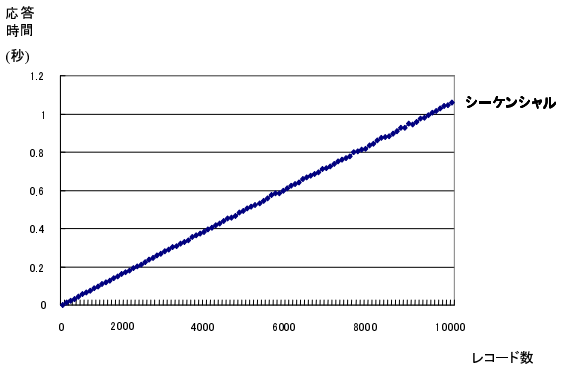

図 8　1 レコード処理によるシーケンシャルアクセス

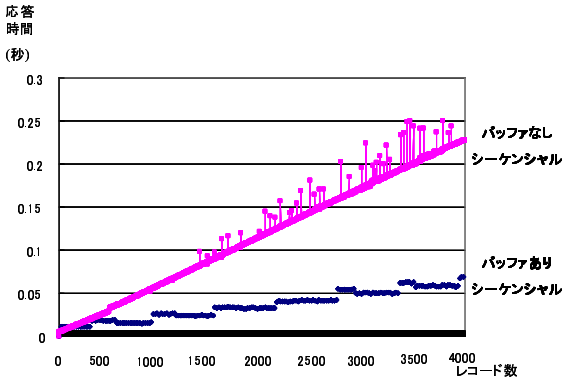

図 9　レコード単位とページ単位のシーケンシャルアクセス

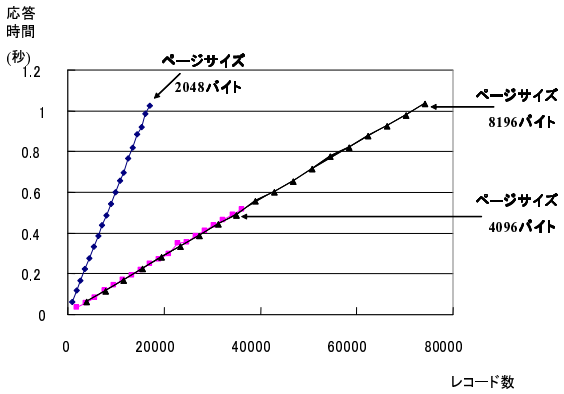

図 10　バッファサイズを変化させた場合の応答時間

次に，バッファサイズを 2KB，4KB，8KB に変化させ，ディスクアクセス時間の比較を行った．結果を図 10 に示す．2KB のバッファを用いた場合，ディスクアクセスのオーバヘッドが発生していると思われる結果が得られた．バッファサイズを 4KB と 8KB に変化させた場合，応答時間に違いは見られなかった．したがってメモリ上に蓄えるデータ量が少ない 4KB のバッファを用いる方が，適当と考えられる．

4.4　検索処理

以上の基礎実験から，4KB のページ単位のシーケンシャルア

クセスが最も効率の良い処理方法ということが判明した．提案した追記型ファイルシステムでは，データ検索に比べ，データ追加処理が頻繁に発生するため，通常，ヘッドはディスクの追記開始トラック上にある．このため，追加処理を実行する場合，平均回転待ち時間で書き込みを行うことが可能である．ヘッド移動が最小限のため，追記処理時間はランダムアクセスによる処理を上回ることはない．

一方，データの検索時は，ディスクサーチが必要となる．データベースシステムでは検索性能を上げるため，一般にインデクスを用いる．提案システムにおいてもインデクスを利用することを提案している．アクセス対象レコードがディスク上に散在し，不要領域が大きくなった場合，シーケンシャルアクセスによる不要データを読み飛ばす時間が，ランダムアクセスによるアクセス時間を上回る可能性がある．その場合，ディスクヘッドの移動によるランダムアクセスが有効である．そのため，3.2 節で提案したインデクスを有効に用いる必要がある．

4.4.1 実験内容

インデクスはディスクに格納されたレコード位置をキーとして張られているため，レコード位置をあらかじめ特定することができる．各レコードの格納場所を，アドレス順にソートすることでディスクアクセスを行う．アクセス対象レコードをアドレス順に検索するため，ヘッドの移動を最小限に抑えて検索をおこなう事が可能である．また，読み込みレコード間隔はインデクスをもとに求めることができ，レコードの間隔が広く，シーケンシャルアクセスによる読み飛ばしに要する時間がランダムアクセスによるヘッド移動時間を上回る場合，ランダムアクセスを採用する．カタログ上の性能では，アクセス時間は 20.33ms であるが，基礎実験 (2) から，本実験でのアクセス時間は 19.84ms ということが判明している．19.84ms 内にシーケンシャルアクセスにより読み取ることのできるデータ量は約 112KB で，208 バイトのレコードでは約 600 レコードにあたる．すなわち読み飛ばし領域が 112KB 以上か 600 レコード以上の場合，ランダムアクセスを利用することが有効である．

全レコード数を 1,000,000 レコードとして 300 回の検索処理を行い応答時間の測定を行った．必要なレコードの位置を，1 番目のレコードは 1，2 番目のレコードは 4，3 番目のレコードは 9，n 番目のレコードは n^2 となるよう配置し検索を行った．つまり，ディスクの検索の初期段階では，必要レコードが密に存在することになり，シーケンシャルアクセスによる読み飛ばしが利用され，必要レコードが疎になる後半部ではヘッドの移動によるランダムアクセスが行われる．

4.4.2 検証結果

図 11 に示すように，すべてシーケンシャルアクセスは，初期段階ではレコード間隔が狭いため，ランダムアクセスに比べアクセス時間が早くなる．レコード間隔は徐々に広がるため，応答時間が二次関数的に増加する．すべてランダムアクセスを行った場合，ヘッドの移動が発生するため，アクセス時間が掛かるが，不要レコード間隔が広がるとシーケンシャルアクセスに比べ，ランダムアクセスによりディスクヘッドを移動させたほうが目的レコードに到達する時間が早くなる．その分岐点は約 230 レコードとなりレコード間隔は約 460 レコードである．ページ単位アクセスでは約 24 ページ目となり，24 ページ目からランダムアクセスを利用する方が良いことがグラフから読み取れる．レコード数をさらに増やした場合，24 ページ目以降はランダムアクセスのみの検索となる．ランダムアクセスの場合も，レコード間隔が広がっていくとディスクヘッドを磁気ディスクの目的の場所まで動かす時間は増加するが，シーケンシャルアクセスによる読み飛ばしによる時間の増加に比べ，十分小さいため，有効な手法である．

以上の実験から，ランダムアクセスとシーケンシャルアクセスを使い分けることで常に高速なディスクアクセスを実現することが可能となる．個人利用のシステムであるため，読み飛ばし領域の広い検索処理が発生する機会は少ないと考えられるため，提案システムは有効であると考える．また，レコードサイズが小さい場合，シーケンシャルアクセスによる読み飛ばすレコード数が増加するため，より高速にデータベース処理を行うことが可能となる．

また，モバイル機器で小型ディスクドライブを利用するにあたり，消費電力の問題を考慮する必要がある．ディスクドライブが電力を消費する動作として，ヘッドの移動，ディスクの起動が上げられる．提案システムでは，ディスクをスケジュールしてからアクセスするため，ヘッドの移動は最小限に抑えることが可能である．また，小型ディスクドライブはディスクアクセスが行われない時，消費電力を抑えるためディスクの回転を停止する．モバイル機器で小型ディスクを利用した場合，数秒おきに GPS データなどが追加されるため，ディスクの停止と起動を繰り返し，ディスクにトルクを与えるための電力を消費する．提案システムでは，頻繁に到着するデータを記録できるようディスクは常に回転しており，停止や起動が起こらない．したがって，上記 2 点の理由から，消費電力の面においても，提案システムは有効である．

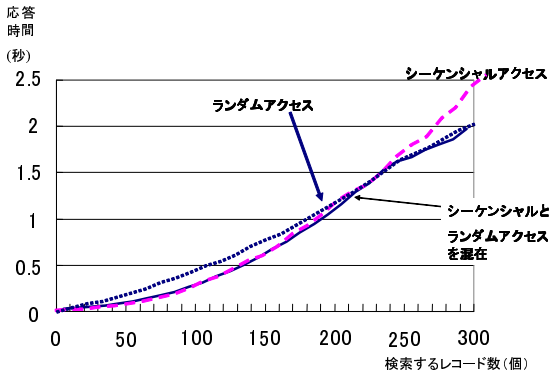

図 11 検索処理

5. 関連研究

以下に関連研究を列挙する．

（1） **スマートカード上でのデータベース**

Bobineau らは PicoDBMS のなかで 1MB を超える非接触型スマートカード上で効率よく問合せを処理する RDBMS について論じている [6]．スマートメディア上でのデータベースは，既存のリレーショナルデータベースやファイルシステムの拡張であるといえる．

提案システムでは，PicoDB のリングモデルとは異なる．PicoDB はスマートカードのみを対象としているが，我々のシステムでは，ディスクを対象としている．また，データを連続的

に書き込むため，コンパクトフラッシュでひとつのデータブロックにアクセスが集中するといった問題を回避できることから，コンパクトフラッシュにも応用が可能である．

倉光らは，非接触型スマートカード上に小型の DBMS を設計，構築する方法について論じている [11]．非接触型スマートカードは記憶容量が小さいため，認証されたホストとカードの上で分散的に DBMS を構築する手法を提案している．利用者は非接触型カード上のストレージで最低限のデータを持ち歩く．

提案システムでは，十分にストレージ容量が大きいため，データベース管理システム全体とデータをストレージ内に収めることができ，小型ディスクドライブ内でクローズしたシステムとなる．

（2） **移動体計算環境における能動型データベース**

寺田らは無線通信環境を有する移動体計算環境での，データベースの接続，切断，データ交換などをルールで処理する AMDS(Active Mobile Database System) について提案・実装を行っている [13].

提案システムでは，ルールを分散データベースの制御に利用するのではなく，能動型データベースの知識処理に利用する．また，能動型データベース管理システムを小型ストレージ上に構築し，統一インターフェースを提供することにより，対応モバイル機器を選ばないポータブルなデータベースシステムが構築できる．

（3） **フラッシュメモリファイルシステム**

石墨らはフラッシュメモリ上でのデータベース構築手法について述べている [10]．フラッシュメモリは記録済み部分への上書きができず，いったんブロックごとにデータを削除する必要がある．削除時間は約 1 秒程度と計算機から見ると非常に長い時間を必要とする．また，メモリブロック消去回数に制限があるため，リンクリストを用いることで特定のブロックに消去が集中することを回避する手法を提案している．

提案システムでは，マイクロドライブ上に追記型ストレージモデルを構築することも提案している．このシステムはフラッシュメモリにも適用が可能である．これにより，ブロック削除回数制限を回避することができる．

（4） **自律ディスク**

横田らはディスク装置の制御用プロセッサやキャッシュメモリを利用することでストレージ機器内で負荷分散，障害対策，障害回復を実現する手法を提案している [9].

提案システムでは，ストレージ上のリソースを利用するという点は類似しているが，制御用プロセッサをディスククラスタの管理に利用するのではなく，データベースの処理に利用する点で異なっている．

（5） **Log-Structured File system**

Rosenblum は，ファイルシステムとしてログ型の記憶モデルを利用することを提案している [8]．ファイル削除が実行されると，ストレージに断片的に未使用空間が作成される．ストレージを効率的に利用するため，随時ガベージコレクションを実施する．

提案システムでは，過去から現在までの情報を検索するテンポラルクエリの実行を想定しているため，データの削除を行わない．これにより未使用空間が分散的に発生することは無く，ガベージコレクションは行わない．

6. おわりに

小型ディスクドライブ上にログを利用した追記型ファイルシステムを構築することで高速にデータアクセスが可能なデータベースシステムの構築が可能となる．また，ディスク内リソースを利用することで，動作機種に依存しないデータベースを構築することが可能となる．

本稿では，データ複製方式ついて検証を行ったが，差分方式の検証も行う必要がある．今後，ライフログを記録しながら問合せ処理を行う際の性能の検証を行う予定である．さらに，時系列のログファイルシステムの特性を利用したテンポラルクエリを実行し，その応答時間の検証も行っていきたい．

謝　　辞

本研究の一部は，科学技術振興機構戦略的創造研究推進事業 (CREST)「高度メディア社会の生活情報技術」プログラム，日本学術振興会科学研究費補助金（課題番号:15300029,15700090），情報ストレージ推進機構 (SRC) の支援によるものである．ここに記して謝意を表す．

文　　献

[1] HITACHI サーバ向ハードディスクドライブ 製品仕様 Ultrastar 15K73 シリーズ. http://www.hgst.com/japanese/products/ultrastar/files/15k73-scsi.pdf.

[2] HITACHI マイクロドライブ 製品仕様 3K4 シリーズ. http://www.hgst.com/japanese/products/microdrive/files/3k4.pdf.

[3] HITACHI マイクロドライブ 製品仕様 DSCM シリーズ. http://www.hgst.com/japanese/products/microdrive/files/dscm.pdf.

[4] MPEG-4 Overview . http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.

[5] SDGPS SkyFix XP. http://www.thales-geosolutions.com/Thales_geo/skyfixxp/skyfixxp_home.cfm.

[6] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDBMS: Scaling down Database Techniques for the Smartcard. *International Conference on Very Large Databases*, No. 27, pp. 11–20, 2000.

[7] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio File Cache: Surviving Operating System Crashes. In *7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 74–83, 1996.

[8] M. Rosenblum. *The Design and Implementation of a Log-Structured File System*.

[9] H. Yokota. Autonomous Disks for Advanced Database Applications. In *International Symposium on Database Applications in Non-Traditional Environments*, pp. 441–448, November 1999.

[10] 石墨紀孝, 最所圭三, 福田晃. 組み込みシステム向けフラッシュメモリファイルシステムの設計 s. 電子情報通信学会論文誌, Vol. J84-D-I, No. 1, pp. 90–99, 2001.

[11] 倉光君朗, 坂村健. 非接触型スマートカード上のユビキタスデータベース. 情報処理学会論文誌 : データベース, Vol. 43, No. SIG5(TOD14), pp. 110–117, 2002.

[12] 小林大, 山口宗慶, 花井知広, 渡邊明嗣, 田口亮, 林直人, 上原年博, 横田治夫. 仮想ノードを用いた自律ディスクシステム更新の高可用化. 電子情報通信学会 信学技法, Vol. DE2003, No. 109, pp. 39–44, 2003.

[13] 寺田努, 塚本昌彦, 西尾章治郎. 移動体計算環境におけるアクティブデータベースの動的トリガグラフ構築機構の設計と実装. 情報処理学会論文誌 : データベース, Vol. 43, No. SIG12(TOD16), pp. 52–63, 2002.