統合化ファイルアクセス法のCD-ROMへの適用*

2G－8

古舘 丈裕　鈴鹿 豊明†
日立ソフトウェアエンジニアリング(株)‡

1 はじめに

マルチメディアデータを取り扱うアプリケーションが増加し、盛んにCD-ROMが利用されるようになってきた。我々はマルチメディアデータベースの研究の一環として、大容量データへの効率的アクセスと柔軟なキー検索の実現を狙いとした統合化ファイルアクセス法(IFAM)[1]を考案、試作してきた。IFAMは抽象データ型として定義し、そのファイル構成およびファイルやレコードへの操作系は論理性の高いものになっている。しかし、当初の実装をハードディスク上での利用を前提として行ったため、そのままの構造ではシーク速度の遅いCD-ROM上ではあまり良い性能が得られなかった。そこで、CD-ROM媒体の特性を踏まえてIFAMの物理的実装方法を改良した。本稿では、CD-ROM上のデータへの効率的なアクセス、特にキーワードによる全件検索における処理速度の向上を実現する一手法について述べる。

2 IFAMのファイル構成

2.1 論理構成

最初にIFAMのファイル構成について簡単に述べる。構成要素としては、(1)データの最少単位(1バイトに対応)であるアトム、(2)アトムの順序列であるレコード、(3)レコードの0個以上の集まりであるファイル、(4)レコードに付けることのできる任意の値であるタグ、がある。基本的に従来のレコード型アクセス法に似ているが、次のような特徴を持っている。

- レコードの長さは一定でなくてもよく、無限可変長である。
- タグとレコードとの関係はm:nにすることができる。また、タグは分類することができ、各々の分類はタグクラスとして定義する。

レコードの長さに制限が無いため従来の数値や文字に加えて、音声や画像など任意のデータをレコードとして格納することができる。また、タグは複数のレコードに付けることができるので、ある特定のタグあるいは範囲指定タグの検索により複数のレコードを扱える。

2.2 内部データ構造

タグとレコードとを関連づけるインデックスはB^+-treeを用いて構築している。リーフノード内には図1のように、タグとレコードポインタの組が配列状に並んでいる。検索操作には主にこのB^+-treeが使用される。すなわちこの部分の実装方法が検索処理の性能を左右することになる。

ところで、HD用のIFAMではLRU法を用いたバッファリングを行いI/Oの効率を上げている。1回のI/Oは1KBのブロックを単位としており、内部データ構造はブロック単位で構成するとバッファリングの効果が出やすい。B^+-treeも1ノードを1ブロックとして構築し、トレースの効率を上げている。

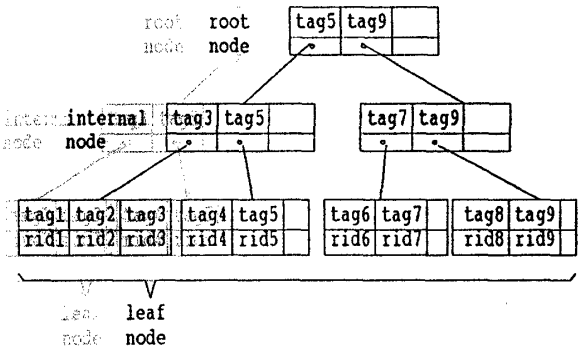

図1: B^+-treeの例

3 CD-ROMへの適用

3.1 処理効率向上のための要件

前述のように、CD-ROMはシークに時間がかかるため、アプリケーション全体の性能を上げるには、いかにしてアクセス回数を少なくするかが重要になる。メ

*Apply to CD-ROM on Integrated File Access Method
†Takehiro Furudate, Toyoaki Suzuka
‡Hitachi Software Engineering Co., Ltd.

モリへのアクセスタイムがnsec単位であるのに対し、ハードディスクへのアクセスにかかるシークタイムは数msec、CD-ROMでは数百msecにもなる。全件検索など2次記憶へのアクセスが大量に発生するような処理では、CD-ROMでは処理時間が大きくなり過ぎ致命的である。

3.2 冗長領域の削除による効果

B^+-treeは、インデックスの構築の一般的な手法の一つであり、元々、要素の挿入や削除を効率的に行えるように工夫されたデータ構造である。しかし、ノードの大きさをブロックサイズで規定する以上、ブロック内には未使用の領域が存在することになってしまう。一般的にはノードの平均充填率は75％である。

B^+-treeでも理想的にはノードを可変長にアロケートした方がスペース効率が良くなるが、アロケーションアルゴリズムが複雑になり、オーバーヘッドとなる。しかし、更新のないCD-ROMではその問題が起きないため、可変長ノードを採用することができ、充填率100％の木を構築することができる。全リーフノードへアクセスするような処理では、充填率が25％アップすれば、単純に考えてアクセス回数の25％削減が見込める。

3.3 データの配置方法による効果

HD用IFAMでは、前述のように各データ構造はブロック単位で構成されるので、更新処理によるブロックの追加・削除が頻繁に起こると、データの内容はディスク上にランダムに配置されることになる。

CD用のIFAMでは、HD用のIFAMのバッファリング機構を利用するためにI/Oの単位はブロックサイズとする。しかし、データの構成については、その大きさが変化することがないためブロックサイズにとらわれずに自由に構築することができる。すなわち、全ての内部データ構造は連続領域に順番に詰めていき、その先頭からの位置でそれらを管理することが可能になる。I/Oは内部データ構造の境界（レコードの境界、B^+-treeのノードの境界等）を気にせずに行うことができる。（図2）

また、関連するデータを連続領域に格納することができるので、続けて参照されるデータが近い位置にあれば、1回のI/Oで同時に読み込まれる可能性が高くなり、アクセス回数の減少が期待できる。

3.4 B^+-treeの実装方法の改良

これまで述べた方法を生かしてB^+-treeのノードの配置を工夫することにより、タグによる全件検索の効率を上げることが可能となる。これは、中間一致すなわち、タグの一部を指定してそれを含むタグのついたレコードを検索するというような処理であり、指定した値が含まれているかどうか全てのタグを調べなければならない。

B^+-treeで構築したインデックスでは全てのタグはリーフノードに格納されている。CD用IFAMでは、図2に示すように、これらのリーフノードをCDの連続領域に格納する。こうすることにより、全件検索では全てのリーフノードを順に読み込んでいけば良く、少ないシークで全てのリーフノードを読み込むことができるようになる。

1～30バイト程度のタグが20万件入っているデータベースで、IFAMを用いてタグの中間一致全件検索を行なったところ、10秒程度で処理が完了した。

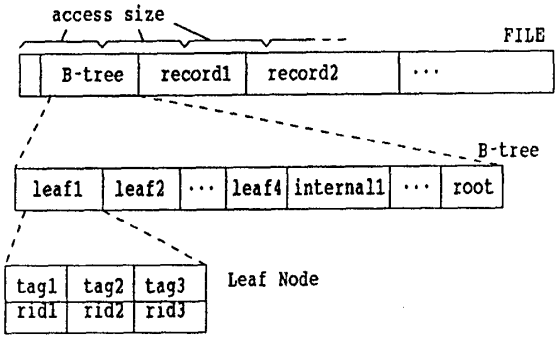

図 2: CD-ROMへのB^+-treeの格納形式

4 まとめ

IFAMの論理的構成は変えずに、CD-ROM上への物理的実装方法を改良することにより、IFAMを用いたCD-ROM上のデータへのアクセスが効率良く行えるようになった。特にタグの全件検索は、HDのファイルをCDにそのまま移行した場合に比べて、1リーフノードにアクセスする毎に発生していたシークを1回に減少させたため、劇的に高速化することができた。

参考文献

[1] 盛屋, 鈴鹿: 統合化環境を目指したファイルアクセス法の提案, 情報処理学会第44回全国大会, 1992.