5P−2

統合化ファイルアクセス法(IFAM)の実現手法*

古舘 丈裕　鈴鹿 豊明　吉田 晶子　盛屋 邦彦†
日立ソフトウェアエンジニアリング（株）‡

1　はじめに

最近ますます需要の増加してきたマルチメディアデータを扱うことのできるデータベースを構築するにあたり、その基礎となるファイルアクセス法に目を向けると、従来のアクセス法にはいくつかの構造／操作上の制約があり、それが性能向上の妨げとなるとともにアプリケーションを作りにくくしている。よって、マルチメディアデータを効率良く操作できる機能を持ったアクセス法が必要となるが、従来のアクセス法に対応した膨大なアプリケーションを無視することはできない。新しいアクセス法としては、マルチメディアに対応する新しい機能とともに、従来のアクセス法との互換性を持ち、旧アプリケーションをそのまま使用できることが望ましい。

そこで統合化ファイルアクセス法（IFAM）を提案した。これは、既存のファイルアクセス法の構造／機能を包含するとともに、無制限長レコードやマルチキーアクセス、レコード内操作などの機能を持ち、新しいアプリケーションにも対応できるものである。本稿ではこの統合化ファイルアクセス法を実現する際の具体的手法について述べる。

2　統合化ファイルアクセス法（IFAM）の概要

従来のファイルアクセス法はその性質からストリーム型とランダムアクセス型の2つに大別されると考えられる。前者はそれ以上分けられないデータ要素（バイト、ワード等）を単位に、要素の順序位置をもとにしたアクセスであり、後者はいわゆるレコードをアクセス単位とし、レコードに付けられたキーをもとにしたアクセスである。IFAMのファイル構成はこれらの両方に対応できるようなものになっている。すなわち、

1. 最小のデータ要素であるアトム（バイト、ワード等）
2. 全順序関係を持ったアトムの並びであるレコード
3. 任意個のレコードに任意個付けられるタグ
4. レコードの集まりであるファイル

となっている。構造的にはランダムアクセス型のファイル構造に近いが、レコードの中身がアトム列となっており従来のストリーム型ファイルに対応できるようになっている。

IFAMの特徴として、従来のランダムアクセス法と違って同じタグを複数のレコードに付けることができ、また1つのレコードに複数のタグを付けることができる。ここでキーと呼ばずタグと呼ぶのは、キーとは元来レコードを一意に識別するものであるが、1つのキーを複数のレコードに付けることを許すと、キーは1つのレコードではなく同じキーの付いたレコード群を指し示すことになり、本来の意味と異なってくるからである。このようにIFAMは柔軟なインデックス構造をもっている。

またIFAMは上記のファイル構造に対する論理的な操作を提供しており、それらはおおまかにファイル単位、レコード単位、アトム単位の操作に分類される。IFAMのもう1つの特徴はアトム単位操作、すなわちレコード内操作を提供していることである。既存のランダムアクセス法ではレコードが最小単位であり、アプリケーションとそのレベルでのやりとりしか許されておらず、またその大きさの上限はOSに規定されていた。そのため画像、音声等の巨大データをひとつのレコードとした場合、その一部分を編集することは効率が悪く、場合によっては不可能であったが、レコード内操作によりそれを回避することができる。また、既存アクセス法にはなかった、途中位置でのデータ（アトム）の挿入・削除も可能とすると共にレコードの長さは論理的に無限長とした。

3　IFAMの実現手法

以上述べてきたIFAMの構成は、大きく分けるとタグとレコードとの間のm:nの関係を実現するタグ管理部と、無限可変長レコードを実現するレコード管理部の各モジュールに分けられる。

3.1　タグ管理部

タグ管理部について、タグを格納するデータ構造にはインデックスの一般的実現手法であるB^+-treeを採用した。一般のB^+-treeではリーフノードにレコードの実体が入るが、ここではそのタグをもつレコードへのポインタが入る。このことにより、レコードの大きさがB^+-treeのノードの大きさに規定されなくなるとともに、複数のタグが1つのレコードを指し示すことが可能となる。しかし、一方で1つのタグから複数のレコードを指すことはできない。そこでタグからのポインタはレコード群を指すようにした。レコード群とは同じタグの付いたレコードの集まりであり、具体的には各レコードへのポインタの集合として表現される。その各ポインタの格納法にはやはりB^+-treeを用いた。それぞれのタグについて、そのタグの付く各レコードへのポインタを格納したB^+-treeが生成される。すなわち、タグを格納するB^+-treeのリーフノードには、そのタグがただ

*Implementation of Integrated File Access Method
†Takehiro Furudate, Toyoaki Suzuka, Akiko Yoshida, Kunihiko Moriya
‡Hitachi Software Engineering Co., Ltd.

一つのレコードに付いているならばそのレコードへのポインタが、複数のレコードに付いているならばそれらのレコードへのポインタを格納した B^+-tree へのポインタが入る。図 1はこの状態を表したものである。これにより、タグとレコード間の m:n の関係が実現されている。(a) はレコードにタグを付加するという一般的イメージを表したもので、(b) はそれを表現する具体的なデータ構造を表したものである。図中の RID とはレコードの識別子であり、レコードへのポインタの役割を果たしている。この例では、dome というタグは 45 番、48 番、76 番のレコードに付いていることが分かる。また 45 番のレコードには baseball、dome、ticket の 3 つのタグが付いている。

効率面を見ると、B^+-tree を用いていることより、タグによるレコードの検索、タグの追加・削除は全タグ数 N に対して log N のオーダの計算量となる。

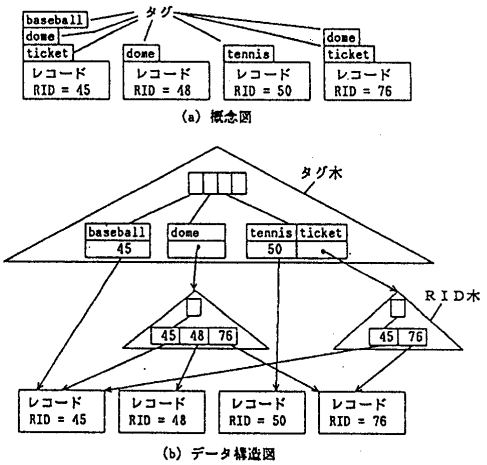

図 1: m:n の関係のタグとレコード

3.2 レコード管理部

レコード管理部について、レコードは無限可変長でしかも途中位置への要素の挿入削除を可能としている。これを効率良く実現する手法として OB-tree[STON84] を用いた。これは、B^+-tree のリーフノードに逐次的に要素を蓄積し、中間ノードにはキーの代わりに自分の子部分木中の要素数（これを重みという）を持たせる手法である。

この OB-tree には、「レコードを指定してそのタグを得る」という逆参照のために、そのレコードに付いている全てのタグを格納している。タグの検索を高速にするためタグはソートして格納するのであるが、タグの長さは可変長であるため、各タグにつき固定長のスロットを用意しスロットの方をソートして OB-tree の先頭に格納するようにした。各スロットには対応するタグの実体の長さと位置が格納される。スロットに続いてタグの実体が入力順に格納され、その後に実際のレコードデータが格納される。すなわち、1 つのレコードは OB-tree の内部を、タグの順序を表現したスロット領域、タグの実体を格納したタグ領域、レコードの内容が入ったレコード領域の 3 つの部分に分けて表現される。これらの位置関係を表した OB-tree の例を図 2に示す。この例では、アトムの大きさは 1 バイトであり、図中の数値はバイト数を表す。例えば baseball というタグは全タグ中アルファベット順で最小のものであるから、そのスロットはスロット領域の先頭に位置されている。そのスロット内には baseball の OB-tree 内の先頭からの位置 30 と長さ 8 が保存されている。

実際のシステムではレコード内部を表す OB-tree とレコードの管理情報を格納した管理ブロックの組が 1 つのレコードを実現する。管理情報には OB-tree へのポインタ、そのレコードに付加されているタグの数、OB-tree 内のレコードデータの始まる位置等がある。

また、OB-tree を用いることでレコードの途中位置へのデータの挿入削除が容易に行える。レコード内操作の計算量はレコード内の全アトム数 N に対して log N のオーダになっている。

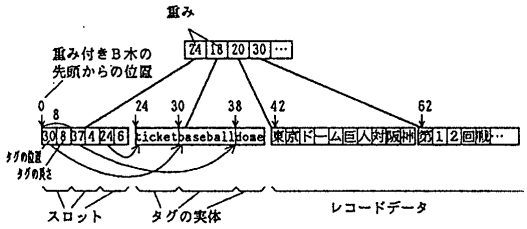

図 2: レコードの内部構造

3.3 ファイルのデータ構造

一つのタグはタグ管理部の B^+-tree とレコード管理部の OB-tree の両方に格納される。これらを関連させて、タグ－レコード間に互いに参照しあうことのできる m:n の対応関係が実現されている。IFAM では、この互いに関連しあったタグとレコードの集まりを 1 つのファイルとしている。すなわち、図 1の全体で 1 つのファイルを表している。

4 むすび

本稿では、マルチメディアデータに対応するための新しい機能を備え、かつ従来のファイルアクセス法の構造／機能を包含した統合化ファイルアクセス法についてその実現手法を述べた。新しい機能として m:n のキーアクセス構造とレコード内操作を提供し、前者は B^+-tree を複数組み合わせて、後者は OB-tree を用いてそれぞれ実現される。全タグ数 N のときのレコードに関するタグの操作、及びレコードサイズ N のときのレコード内操作の計算量はそれぞれ log N のオーダとなっている。

参考文献

[STON84] M. Stonebraker, L. A. Rowe: Database Portals :A New Application Program Interface, Proc.10th VLDB, 1984.