7N-2

データの検索頻度を考慮した最適なインデックス生成に関する研究

小熊　祐子†，南澤　吉昭 †　吉井　伸一郎†

北海道大学大学院　情報科学研究科†

一般にインデックスはデータベースの検索速度を高速化するために使われるが，アプリケーションからの検索特性を考慮しているわけではない．例えば，頻繁に検索されるデータとあまり検索されないデータのように偏りがある場合，既存のインデックスのデータ構造である BTree では必ずしも最適な高速化がなされるとは限らない．そこで本研究ではデータの検索頻度を考慮したインデックスのデータ構造を提案し，既存のデータ構造と比較を行う．検索頻度の偏りに応じ，木のバランスを変更することによって最適化をはかり，検索頻度まで考慮した平均検索時間の短縮を行う．

１．はじめに

近年，情報技術はめざましい発展をとげており，今まではシステム化されていなかった部分も，次々とシステム化されてきている．それに伴い取り扱うデータ量も増大しており，そのデータを管理するため様々な場面においてデータベースが利用されている．

データ量が増えるにつれ，検索スピードは低下していくが，それを防ぐためにデータベースにはインデックスが存在している．

インデックスとはデータの検索速度を向上させるために，どの行がどこにあるかを示した索引のことである．データ検索時に、目的のデータが見つかるまですべての行を一行ずつ調べていくよりも，索引を利用して目的の行の場所を見つけてからその行のデータを読み取る方が効率的であるという考えに基づき，非常によく用いられている．特に大きなテーブルでは，インデックスを用いることにより，大幅にパフォーマンスが改善される．

代表的なインデックスには BTree インデックス，ハッシュインデックス，ビットマップインデックス[1]などが挙げられるが，最もよく利用されるインデックスは BTree インデックスである．

また BTree インデックスは他のインデックスに比べ，様々なデータベースアプリケーションで実装されている．しかしながら，データの検索頻度を考慮し，木のバランス配分を変更するというインデックスは提案されていない．そこで，本報告ではデータベースの検索頻度を考慮し，木のバランス配分を変更する UN-BTree インデックスを提案する．

「A study on optimal index generation based on search frequency of data」

† Yuko KOGUMA, Hokkaido University
Yoshiaki MINAMISAWA, Hokkaido University
Shinichiro YOSHII, Hokkaido University

２．BTree 構造

BTree 構造は R. Bayer et, al,[2] によって考案されたデータ構造で，ルートから葉までの高さが等しく，一つのノードの中に入っているデータの数が極端にばらつかないように，木のバランスが取れているため，どの値が検索されてもほぼ同じレスポンスタイムを返すという特徴を持つ．BTree はルートノード，ブランチノード，リーフノードという三種類のノードを持つ。ルートノード，ブランチノードは図１のような構造を持ち最大 *m* 個のポインタ，*m-1* 個のキー，最小 *m/2* 個のポインタ，*(m/2)-1* 個のキーを持つ（*P* はポインタ *K* はキーを表す）．これを *m* 階の BTree と呼ぶ．リーフノードは最大 *m* 個，最小 *m/2* 個の RowID へのポインタとキーを持つ．各ノードに平均*(3/4)m* 個のデータが入っている場合の平均比較回数を考えると，キーの比較回数を *a* とした場合，各レベルでの比較回数は *(3/8)ma* となる．ルートノードからリーフノードまでの高さを *h* とすると，目的のデータを見つけるまで平均　*(3/8)ma×(h+1)*ステップかかる．

P_1	K_1	P_2	K_2	…	K_i	P_{i+1}

図１　ノード内の構造

３．UN-BTree 構造を用いたインデックス

ここでは，検索頻度に基づき BTree の木のバランス配分を崩した UN-BTree を提案する．通常の BTree では木のバランスは常に一定であるが，UN-BTree では検索頻度によって，木のバランスが変化する．具体的には，検索頻度の高いリーフノードのデータ（ポインタ・キー）のレベル上げを行い，木のバランスを変化させる．UN-BTree インデックスでは，はじめに BTree インデックスを構築し，各キーの検索頻度を取得する．その検索頻度に基づき UN-BTree インデックス構築を行う．

以下では簡単な例を用いて説明を行う．

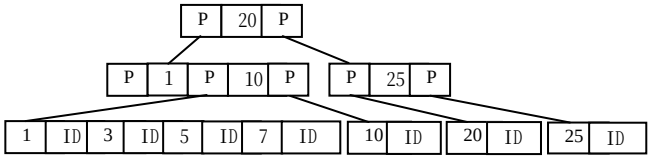

図２　Btree の例

図２において，*P* はポインタ，数値はキーを表す．この一番左のリーフノード内にあるキー値５番を検索頻度の高いキーとする．この状態から UN-BTree を用いて木のバランス配分を変更すると，キー値１番から３番，キー値５番，キー値７番の三つのリーフノードに分割される．そして，キー値５番は検索頻度の高いキーであるため，キー値５番が入っているリーフノードのみ高さが１つ下がり，図３のように現在ブランチノードが入っているレベルに並ぶ.

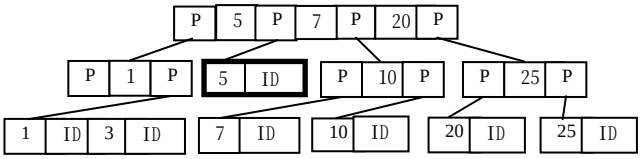

図３　UN-Btree の例

BTreeと比較した場合の検索ステップを以下に示す．目的データまでの検索ステップは元々のリーフノード中のキー位置に従って減少する．リーフノード中にn_{leaf}個の検索キーがあり，目的のキーがi番目にあるとし，またリーフノードの親ノード内にn_{parent}個のキーがあり，リーフノードをさすポインタが第jキーの手前にあったとすると，減少する比較回数は$j+(i-1)+(-1)$である．ここで，iとjは平均するとn_{leaf}とn_{parent}内にランダムに分布すると考えられるので, 平均減少比較回数は

$$\frac{1}{n_{parent}}\sum_{j=1}^{n_{parent}}(j)+\frac{1}{n_{leaf}}\sum_{i=1}^{n_{leaf}}(i-1)-1 \quad (1)$$

(1)より $(n_{parent}+1)/2+(n_{leaf}-1)/2-1$ (2)

となる．ここで，木を m 階の木とし，各ノードには平均して $3/4m$ 個のデータが入っているとすると，(2)式より $(3/4)m-2$ 回減少することができる．さらに，キーのレベル上げに伴い，その他のデータ検索も高速化される場合がある．例えば図３中のキー値７がその例である．

(1)式を導出した設定において，$i+d$ 番目の位置にある検索キーに到達するまでの減少ステップ数は$(j-1)+i+(-1)$ $(1 \leqq d \leqq m-i)$　この場合も目的の検索キーは $i+1$ から m まで均等に分布していると考えられるので，平均すると

$$\frac{1}{n_{parent}}\sum_{j=1}^{n_{parent}}(j-1)+\frac{1}{n_{leaf}}\sum_{i=1}^{n_{leaf}}\{\frac{1}{n_{leaf}-i}\sum_{d=1}^{n_{leaf}-i}(i)\}+(-1) \quad (3)$$

(3)より　$(n_{parent}+n_{leaf})/2-1$ (4)

またそれ以外のキーではレベル上げを行った検索キーより手前のポインタから探索されるキーの比較回数は増減なし，検索キーのポインタ以降のポインタから検索されるキーの比較回数は２回増える．レベル上げにかかわるポインタは均一に分布するものと考えると，平均１回増える．

４．考察

本稿で提案した UN-BTree がどのような検索頻度分布の際に有効であるか，以下の３パターンにて考察する

１．平均的に全てのデータが検索される
２．一つのデータのみの検索頻度が高い
３．検索頻度が高いキーが全体の２割

まず１の場合はデータの検索頻度の偏りが少ないため BTree の検索ステップ数とほぼ変わらない．２の場合，BTree の場合は平均で $(3/8)ma\times(h+1)$ ステップかかるが，UN-BTree の場合，検索頻度の高いキーの検索を行うと検索されるキー値のレベルが１つ上がり，なおかつノード中にデータは一つであるため $(3/8)ma\times(h-1)$ となり比較回数は減少する．ここでは，(4)式の減少分はごく僅かであるため無視した．３の場合，$k=500,h=2$ とし表１の値で比較を行うと，BTree の場合は平均 565a，UN-BTree の場合は平均約 190a になり，UN-BTree は BTree の約 1/3 の比較回数で検索を行うことができる．

	キー数比率	アクセス比率
検索頻度の高いキー	0.2	0.8
その他	0.8	0.2

表１　検索頻度比率表

５．おわりに

本報告ではデータベースへの検索頻度を考慮し，木のバランス配分を変更する UN-BTree インデックスを提案した．既存の BTree に比べ，検索頻度に偏りがある場合は検索速度が速くなることを示した．しかしながら，今回は理論値のみの計算であるため，今後の課題としてベンチマークを行うことが必要である．ベンチマークの方法として、オープンソース DB に UN-BTree を組み込むこと等が考えられる。

参考文献
[1]荒谷聡，森側真一　基礎からわかるデータベース構築ガイド　日経BP社 2002
[2] R. Bayer and M. Schkolnick. Concurrency of Operations on B-trees. In *Acta Informatica*, Vol. 9, pp. 1–21, 1977.