

パーソナル通信用アドレス管理データベースの検討

齊藤 典明 萩原 正敏 春田 勝彦

NTT通信網総合研究所

パーソナル通信は、"いつでも、どこでも、だれとでも"という通信環境を提供するものである。この為従来の端末番号の他に個人番号を導入し、ネットワーク内のDBにおいて個人番号から端末番号に変換する。このDBアクセスには(1)DBは分散配置されユーザ毎にホーム局が定められる。(2)個人番号からホーム局が特定できない。(3)番号変換は移動先の局で行なう。(4)接続遅延を防ぐ為にDB応答時間を一定値以下に抑える。という条件を課す。本論文では、DBのアクセス手法としてダイナミックハッシュの一種である拡張ハッシュを導入することを検討し、改良する事により応答時間を一定値で抑える改良方式を提案する。

A Study on An Address Management Database System for Personal Telecommunication Services

Noriaki SAITO Masatoshi OGIWARA Katsuhiko HARUTA

NTT Telecommunication Networks Laboratories

1-2356 Take Yokosuka-City Kanagawa, 238-03, JAPAN

Personal telecommunication services enable users to communicate with whomever, wherever and whenever. Authors introduced an individual id-number (IID) for each user in addition to an existing telephone number (LID). When a network makes connection, it translates from an IID to an LID using its own database. The translation time have the upper limit concerning connection quality. This paper propose a database accessing method using extendible hashing to keep connection time under the limit.

1. はじめに

パーソナル通信サービス[1][2]とは，“いつでも、どこでも、だれとでも”通信を行なう事を可能とするものである。このサービスを実現するために従来の端末ごとの識別番号（以降LID:Logical Identification と略す）の他にユーザを一意に識別するための番号（以降IID:Individual Identification と略す）をネットワークに導入し、ネットワークがユーザの現在地、スケジュール、指定されている代理人などを把握し端末に入力されたIIDから適切なLIDに変換し呼接続する。このIIDからLIDに変換するために用いられるデータベースをアドレス管理データベース（以降AD-DBと略す）と呼ぶことにし、これはインテリジェントネットワーク[3][4]上の網サービス制御局内に置かれる。

本稿では、AD-DBについて次のような前提条件で検討するものとする。

(1) パーソナル通信サービスのユーザは、21世紀前半において国内だけでも4000万人にも及ぶと考えられ1つのデータベースで対応するにはその規模から見て非現実的であり、AD-DBは分散制御方式で利用することになると仮定する。このときユーザ毎にホーム局を定める。

(2) ホーム局は、ユーザの利用頻度の高いロケーションとする。IIDはユーザによる定義可能、ホーム局を変更しても同じIIDが利用可能という設定にすることからIIDからホーム局が特定できない。

(3) IIDからホーム局が特定できないため呼接続で生じるDBアクセスは、アクセスしたAD-DBにIIDが登録されているとは限らないため不成功探索及び問い合わせ処理が必要な場合がある。これをできるだけ少なくするため番号変換は移動先で行なうことにする。このためユーザの移動により移動先の局のAD-DBは、ユーザを登録及び管理するためにファイルメンテナンスが必要となる。

(4) 評価については、接続制御用データベースのため応答時間を対象とし接続遅延を抑える事を重視する[5]。

これらの条件を前提にデータベースのアクセス手法を選択する。一般にデータベースのアクセス手法から見たデータ構造は大きくわけて逐次構造、木構造、ランダムな構造に分類できる[6]。アクセス時間をキーが入力されてから目的のレコードへ到達するまでの時間とすると平均アクセス時間を最も短縮可能なものは、一般ランダムな構造で用いるハッシュ法である[7][8][9]。また、アクセス時間の上限を最長アクセス時間と定義する

と、逐次構造ではデータ数に比例し、木構造ではデータ数の対数に比例して長くなり、ランダムな構造では逐次構造に近づく事が知られている。本研究では、ユーザが移動し移動先の局でアドレス変換を行なうため移動先の局ではデータ数が変動する。そこでAD-DBのアクセス手法としてデータ数の変動に応じてファイルを分割できる動的ハッシュ法[10]の適用を考えた。

動的ハッシュ法には、衝突した部分のファイルを分割する拡張ハッシュ法と、衝突がある一定以上生じたとき衝突部分とは関係なくファイルを分割するリニアハッシュ法がある。両方式のうちリニアハッシュ法はファイル分割を各ファイル均等に行なうため、キー分布に偏りがある場合や、最長アクセス時間の短縮という目的に対しては不向きであるため考察の対象から外した。

したがって本研究では、想定しているパーソナル通信サービスに拡張ハッシュ法を適用したとき不成功探索により生じる最長アクセス時間及びデータ数の変動に伴うファイルメンテナンス時間の短縮が課題となる。

これに対し従来は主にキー分布が固定された場合の平均アクセス時間の短縮が研究されていたが、本研究では拡張ハッシュ法をベースにデータ数に依存せずキー分布が変動した場合でも応答時間を一定値で抑えることを可能とする改良方式を提案する。

2. 拡張ハッシュ法の概要と問題点

2.1 拡張ハッシュ法の概要

本方式の基本となる、拡張ハッシュ法について簡単に説明する。拡張ハッシュ法はFagin[11]が提案したものでそのデータ構造は、ディレクトリ、リーフページの集合及び一時記憶領域から成る（図1）。ディレクトリにはヘッダと可変長のレコード部がある。ヘッダにはディレクトリの全域深さ（以降depと略す）が記述されている。レコード部は2個の容量を持つ2^{dep}個のレコードから成っており、各レコードには、リーフページをポイントするためのポインタとそのリーフページに記録されているデータのエン트리数が記録されている。リーフページはヘッダと固定長のレコード部からなる。ヘッダはそのリーフページの局所深さ（以降local_depと略す）が記録されている。レコード部は、2個以上の容量を持つ2^S個（Sはある固定した値）のレコードから成っており、各レコードにはキーとそのキーに付随する情報が記録

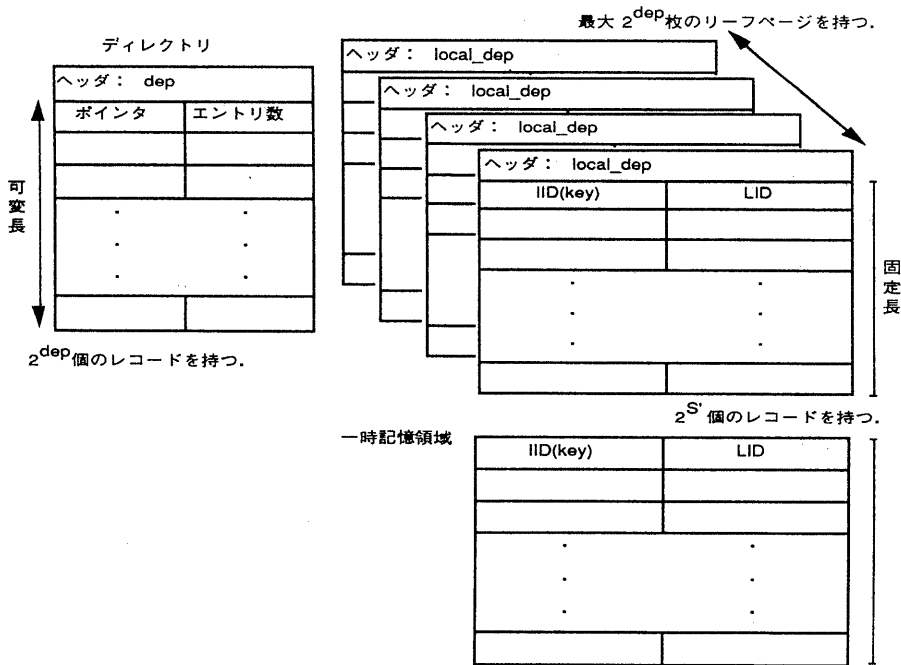


図1. 拡張ハッシュ法のデータ構造

されている。一時記憶領域はリーフページと同じ容量、同じ数だけのレコード部からなり、ヘッダ部分はない。

アクセス方法の概要を次に示す(図2)。

(1) キーにハッシュ関数を施すことにより疑似キーを計算し n ビットの二進表現になおす。

(2) 疑似キーの下位 dep ビット(文献[10]による)を二進木状に分類して行くことにより、木構造の一種のトライを配列で表現した形のディレクトリ内を検索し、ディレクトリ内の該当レコードが指定される。

(3) 該当レコードに記録されているリーフページのポインタを得てリーフページへのアクセスを行なう。

(4) リーフページ内でのアクセスは、疑似キーの残りの数ビットのうち S' ビットを用いたハッシングによって行う。

(5) 指定されたリーフページ内のレコードに既にデータが書き込まれている、或は該当データ以外が記録されている場合のキー衝突に対して、連鎖法あるいは計算法という衝突処理を行なう。

(6) 拡張ハッシュ法において、ディレクトリで指定されたリーフページが一杯で更にデータを書き込む必要がある場合は、新しいリーフページを導入することによってデータ数の増加に動的に対処する。ここでは、ディレクトリの拡張を要する

場合があり、この場合全域深さは1増加する。特に2つに分割した相手側のリーフページを互いにパディページと呼ぶ。

(7) 反対にデータ数が減少し一定値以下になった場合には、2つのリーフページを合併する。ここでは、ディレクトリの縮小を要する場合もあり、この場合全域深さは1減少する。

2. 2 シミュレーションによる評価と考察

拡張ハッシュ法のAD-DBへの適用性を検討する。評価はシミュレーションで行なうこととし次の条件で行なう。

(1) データ構造においてリーフページ内のレコード容量は必要最低限なものを考慮して2個としキーはIIDそれに付随する情報はLIDのみとした。

(2) IIDは、利用者数の考慮及び簡単化という理由から9桁の正整数とした。

(3) キーの性質から、ハッシュ関数の選択により容易に第2次クラスタ(疑似キーが同一となることによって生じる衝突)を解消、又は一定値以下に抑える事ができる。例えば同族なキー(疑似キーが同一となるキーの系列)を1000個以下に抑えるためには、ハッシュ関数として $\text{mod } 1000000$ をとれば良い。リーフページの大きさが同族キーの数より大きければあふれページを必要としない完全ハッシュの状態を作り出す事ができる。シミュレーションにおいてハッシュ関数は、第2次

2進表現された疑似キー
1 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0 1 0 0

ディレクトリの深さだけ読む

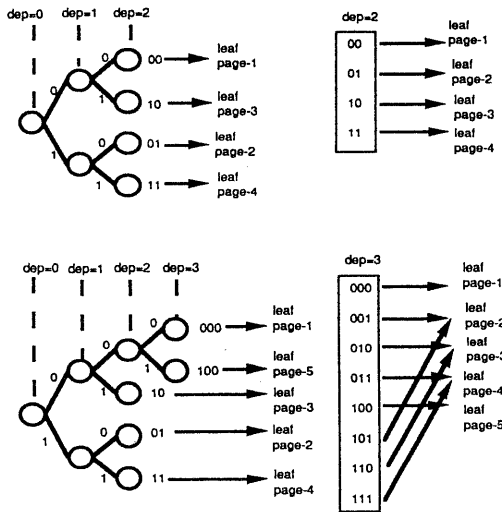


図2. 拡張ハッシュ法のアクセス方法

クラスを一定値に抑えさらに計算時間の短縮のため2の中乗のモジュロ演算（これをシフト演算によって求めた）を用いた。

(4) 衝突の処理は計算法の一種である線型法（衝突したレコードの次のレコードにアクセスをおこなった）を用いた。

(5) 全域深さは最大10程度まで考慮した。

(AD-DB1個あたりのユーザ数を400万人とするとリーフページの長さは4000エントリ程度。)

(6) SUN4を用いオンメモリでシミュレーションを行なった。

拡張ハッシュ法のAD-DBへの適用の妥当性及び問題点を最短アクセス時間、最長アクセス時間及びファイルメンテナンス時間で評価する。

応答時間は、アクセス時間とファイルメンテナンス時間の和とする。アクセス時間は、1つのキーの処理に対して目的のレコードを探しあててまでの時間と、それに必要な操作の時間とする。アクセスの種類として検索、更新、書き込み及び消去があり各々の操作によってアクセス時間は異なる。（但し、LIDの書き込み時間や消去時間、ファイルの分割合併といった操作時間は含まないものとした。）ファイルメンテナンス時間は、ファイルの分割、合併に要する処理時間とする。

またキー分布をここでは全域深さと局所深さの差で計ることとする。キー分布が一様であれば全

域深さと局所深さの差は0に近づき、キー分布が偏れば全域深さと局所深さの差は大きくなる。

衝突処理のないアクセスを最短アクセスと呼ぶことにしアクセス時間の下限とする。最短アクセス時間の測定結果を図3に示す。拡張ハッシュ法の性質上最短アクセスは、メモリ使用率やファイルの長さ（リーフページ単体の長さ）と全域深さで評価できる。しかしAD-DBの使用目的から個々のAD-DBで管理するキー分布は、人間の行動に応じて常に変化する。各々のAD-DBでキー分布状態が異なるため、ハッシュ関数の設定でキー分布に適応するのは困難である。よって接続品質を確保するという理由から任意のキーの分布における最短アクセス時間の上限を知る必要がある。ここでは、疑似キーの系列が $f(N)=2^N \times i(N, i=0,1,2,3,...)$ となるものに集中する場合について考察を行った。拡張ハッシュ法においてこのような疑似キーの系列に対して非常に弱い構造になっていることがわかる。なぜならば、このような系列のデータが書き込まれている場合、ディレクトリは非常に大きく拡張されているが、キーはある特定のリーフページに集中している。この状態で系列以外のキーの書き込み又は消去を行なう場合、全域深さと局所深さの差が問題となる。全域深さと局所深さの差が大きくなるにつれて、データを書き込み又は消去を行なう時のディレクトリ内のエントリ数更新の手間が $2^{dep-local_dep}$ 回であるためアクセス時間が遅くなるからである。

また非常に多くの衝突あるいは不成功探索によりリーフページのレコード全てを探索した場合を最長アクセスと呼ぶことにしアクセス時間の上限とする。最長アクセス時間は、指定されたリーフページのレコード全てを参照するためにリーフページの長さで決まる。AD-DBではリーフページの長さを動的に変動させる事はないので最長アクセス時間は、一定値で抑える事ができる。また図4は、リーフページの長さで最長アクセス時間の関係を表したものである。これらのことから、AD-DBに要求する時間的制限によってリーフページの大きさを選ばばよい。

ファイルのメンテナンス時間について考察を行なう。メンテナンス時間は、以下の項目によって決まる。なお以下で、Accは上限が最長アクセス時間、下限が最短アクセス時間である検索におけるアクセス時間を表す関数とし、Mntをメンテナンス時間、添字付きはその要因を表す関数とした。また $C_i(i=1,2,3,...)$ は定数を表すものとした。

(1) リーフページの分割に要する時間 (Mnt₁)

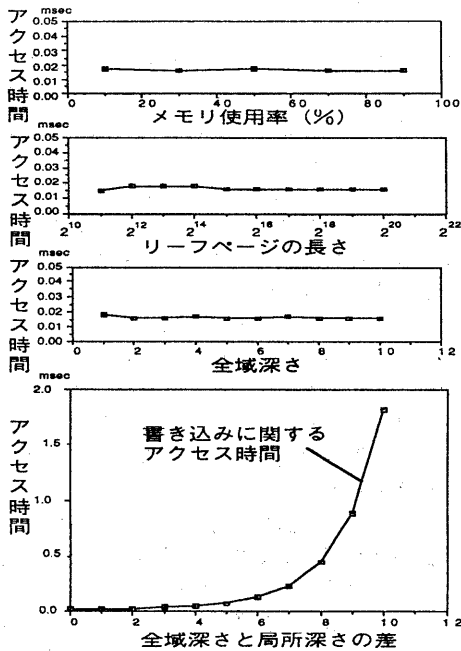


図3. アクセス時間 (下限)

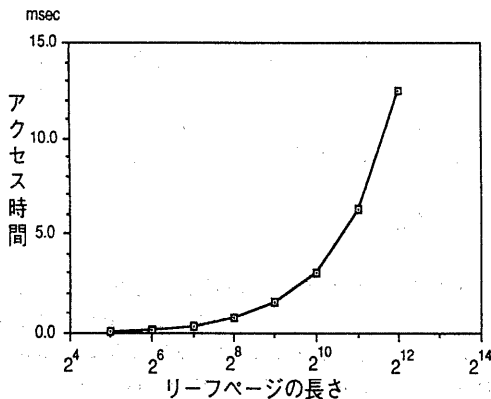


図4. アクセス時間 (上限)

は、その手順 (分割の必要の有無を判定し、必要なら一時記憶領域にデータを移し、1つ1つアクセス手順に従いデータを書き込む。) からアクセス時間と分割時のリーフページ内のエントリ数によって決まる。

$$Mnt_1 = \sum_{i=1}^p Acc \times C_1 + C_2$$

ここでpは該当リーフページのエントリ数である。

(2) リーフページの合併に要する時間 (Mnt_2) は、その手順 (バディページを探し合併の必要の有無を判定し、合併の際は一時記憶領域にデータ

を移し、1つ1つアクセス手順に従いデータを書き込む。) から以下のように表現できる。なおバディページ探索の手間は全域深さと局所深さの差が大きくなるにつれて増える。

$$Mnt_2 = 2^{dep-local_dep} \times C_3 + \sum_{i=1}^p Acc \times C_4 + C_5$$

(3) ディレクトリの拡張に要する時間 (Mnt_3) は、その手順 (拡張の必要の有無を判定し、必要ならディレクトリの長さを倍にする。) から全域深さの深さにつれて遅くなる。

$$Mnt_3 = dep \times C_6 + C_7$$

(4) ディレクトリの縮小に要する時間 (Mnt_4) は、その手順 (縮小の必要の有無を判定し、必要ならディレクトリの長さを半分にする) から全域深さの深さにつれて遅くなる。

$$Mnt_4 = dep \times C_8 + g + C_9$$

ここでgはリーフページ数を変数とする縮小の判定に要する関数としその上限は 2^{dep} の定数倍である。

(5) ディレクトリ内容の更新に要する時間 (Mnt_5) は、その手順 ($2^{dep-local_dep}$ 箇所のディレクトリ内レコードのエントリ数、ポインタを更新し、これを移動したデータの数だけ繰り返す) から全域深さと局所深さの差が大きくなるにつれて遅くなる。

$$Mnt_5 = \sum_{i=1}^p (2^{dep-local_dep} \times C_{10} + C_{11})$$

さらに、キー分布によっては1回のデータの書き込みで複数回リーフページを分割しなくてはならない場合があり、そのときの分割回数nとするとファイル拡張時間、縮小時間は以下のように表すことができる。

ファイル拡張

$$Mnt = \sum_{i=1}^n Mnt_1 + a \times Mnt_3 + \sum_{i=1}^n Mnt_5$$

ファイル縮小

$$Mnt = \sum_{i=1}^n Mnt_2 + a \times Mnt_4 + \sum_{i=1}^n Mnt_5$$

ここで、aはディレクトリの拡張または縮小を伴うとき1で、それ以外は0である。

シミュレーションでは (図5)、Accを最短アクセス時間に設定し、リーフページの大きさは一定値 (2^{10} とした) であることから $\sum Acc$ はある定数値となる。よってグラフの曲線の式は、

$$(a) a=1, n=1, dep-local_dep=0:$$

$$Mnt = Mnt_2 + Mnt_4 + Mnt_5 \\ = dep \times C_{12} + 2^{dep} \times C_{13} + C_{14}$$

$$(b) a=1, n=1, dep-local_dep=0:$$

$$Mnt = Mnt_1 + Mnt_3 + Mnt_5 \\ = dep \times C_{15} + C_{16}$$

(c) $a=0, n=1$:

$$\begin{aligned} \text{Mnt} &= \text{Mnt}_2 + \text{Mnt}_5 \\ &= 2^{\text{dep-local_dep}} \times C_{17} + C_{18} \end{aligned}$$

(d) $a=0, n=1$:

$$\begin{aligned} \text{Mnt} &= \text{Mnt}_1 + \text{Mnt}_5 \\ &= 2^{\text{dep-local_dep}} \times C_{19} + C_{20} \end{aligned}$$

(e) $a=1$:

$$\begin{aligned} \text{Mnt} &= \sum_{i=1}^n \text{Mnt}_1 + \text{Mnt}_3 + \sum_{i=1}^n \text{Mnt}_5 \\ &= n \times C_{21} + \text{dep} \times C_{22} + \sum_{i=1}^n \text{Mnt}_5 + C_{23} \end{aligned}$$

となるはずである。しかしながら、シミュレーション結果からディレクトリ長の伸縮に関しては現在考慮している範囲では変化量が微小であるため(msecのオーダー)一定値であると類推できる($\text{Mnt}_3, \text{Mnt}_4$ 一定)。

(a) $\text{Mnt} = 2^{\text{dep}} \times C_{24} + C_{25}$

(b) $\text{Mnt} = C_{26}$

(c) $\text{Mnt} = 2^{\text{dep-local_dep}} \times C_{27} + C_{28}$

(d) $\text{Mnt} = 2^{\text{dep-local_dep}} \times C_{29} + C_{30}$

(e) $\text{Mnt} = n \times C_{31} + \sum_{i=1}^n \text{Mnt}_5 + C_{32}$

これらの検討結果からわかる通り、キー分布(全域深さと局所深さの差が生じる場合)及びデータ数(ディレクトリの深さが大きくなった場合)の変動によりAD-DBの性能が落ちる場合が存在する。そこで接続遅延を抑えるための改良が必要である。

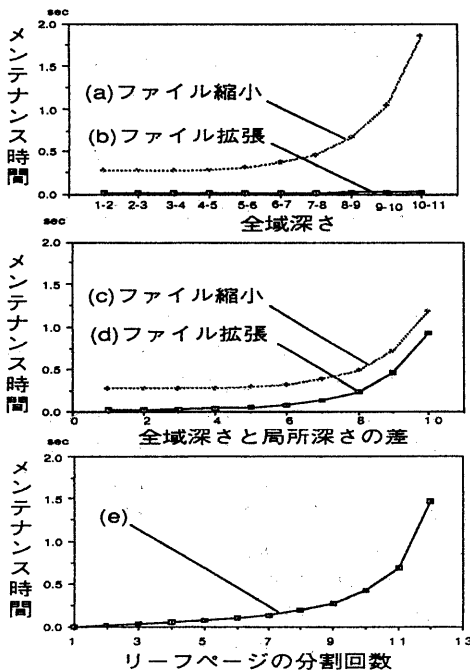


図5. 拡張ハッシュ法のメンテナンス時間

3. 改良方式の提案

これらの問題点を改良するためにデータ構造を変更する(図6)。改良点は以下の4点である。

(1) データの書き込みにおいて、全域深さと局所深さの差によるアクセス時間の劣化を改善するためにディレクトリのデータ構造を変更する。ディレクトリ内の各リーフページのエントリ数の記述を各リーフページ内に持たせることにより、データの書き込み1件あたりエントリ数の更新は丁度1回で完了するようになる。

(2) ファイルの縮小の際に生じるパディページ検索の高速化のために、パディページに関する記述を各リーフページに持たせる。これにより全域深さと局所深さの差が大きくなることによる性能劣化を抑えられる。

(3) ファイルの拡張、縮小の際に用いていた一時記憶領域を不必要であると思われるため消去する。これによりファイルの拡張、縮小が高速になっただけでなく、複数回リーフページを分割するときでさえ一回の分割とはほぼ同じ程度の時間で抑えることができる。(計算時間が若干増えるが、計算時間はデータの移動に比べ無視できる。)

(4) ディレクトリ縮小の判定の高速化のために、ディレクトリ内にリーフページの展開状況の記述を持たせる。これによりディレクトリ縮小時間を一定時間で抑えることができる。

4. 性能比較

最短アクセス時間の評価結果を図7に示す。ディレクトリ内のエントリ数の更新をなくすことにより、最短アクセス時間の下限が指数関数的に増加していたものが定数値で抑えられた。

最長アクセス時間については、特に改良は行っていないので以前のままである。

次にファイルのメンテナンス時間について、先の項目について考察を行なった。この結果キー分布やデータ数の変動がある場合でも、接続遅延を抑えることが可能である事がわかる。

(1) リーフページの分割に要する時間は、その手順(分割の必要の有無を判定し、必要データについてのリーフページ間の移動を行なう。)からアクセス時間と分割時のリーフページ内のエントリ数によって決まる。

$$\text{Mnt}_1 = \sum_{i=1}^p \text{Acc} \times C_1 + C_2$$

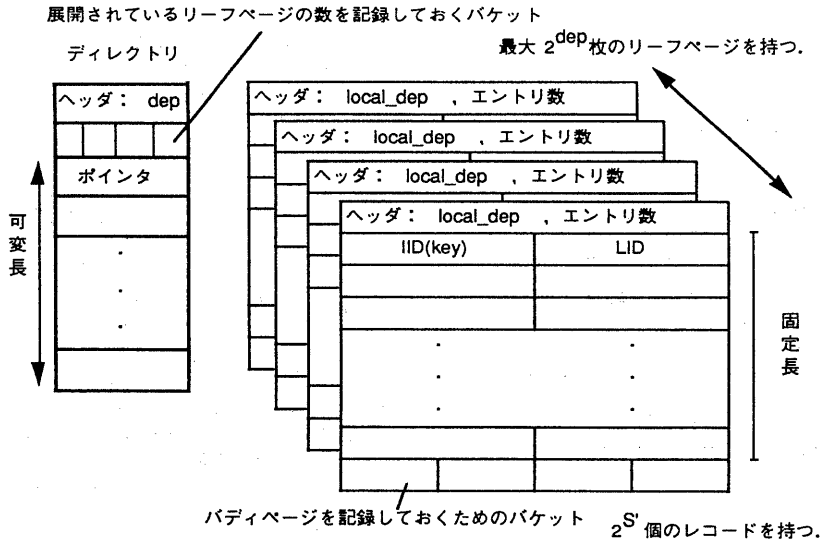


図 6. 改良方式のデータ構造

(2) リーフページの合併に要する時間は、その手順（バディページを探し合併の必要の有無を判定し、合併の際は直接リーフページ間で移動を行なう。）から以下になる。なおバディページ探索の手間は定数値である。

$$Mnt_2 = \sum_{i=1}^p Acc \times C_3 + C_4$$

(3) ディレクトリの拡張に要する時間は、その手順（拡張の必要の有無を判定し、必要ならディレクトリの長さを倍にする。）から全域深さの深さにつれて遅くなる。

$$Mnt_3 = dep \times C_5 + C_6$$

(4) ディレクトリの縮小に要する時間は、その手順（縮小の必要の有無を判定し、必要ならディレクトリの長さを半分にする）から全域深さの深さにつれて遅くなる。

$$Mnt_4 = dep \times C_7 + C_8$$

(5) ディレクトリ内容の更新に要する時間は、その手順（必要箇所を計算しポインタを更新する。）から一定値となる。

$$Mnt_5 = C_9$$

以上のことからシミュレーションにおけるグラフの曲線（図 8）の式は

(f) $a=1, n=1$:

$$Mnt = Mnt_2 + Mnt_4 + Mnt_5 \\ = dep \times C_{10} + C_{11}$$

(g) $a=1, n=1$:

$$Mnt = Mnt_1 + Mnt_3 + Mnt_5 \\ = dep \times C_{12} + C_{13}$$

(h) $a=0, n=1$:

$$Mnt = Mnt_2 + Mnt_5 \\ = C_{14}$$

(i) $a=0, n=1$:

$$Mnt = Mnt_1 + Mnt_5 \\ = C_{15}$$

(j) $a=1$:

$$Mnt = \sum_{i=1}^n Mnt_1 + Mnt_3 + \sum_{i=1}^n Mnt_5 \\ = C_{16} + Mnt_3 + Mnt_5$$

となるはずである。同様に 2. 2 の結果を用いて、ディレクトリ長の伸縮に関しては、 Mnt_3 , Mnt_4 = 一定と見なし変形できる。

(f) $Mnt = C_{17}$

(g) $Mnt = C_{18}$

(h) $Mnt = C_{19}$

(i) $Mnt = C_{20}$

(j) $Mnt = C_{21}$

5. おわりに

本論文では応答時間としてキーが入力されてからデータを格納しているレコードに到達するまでの時間とファイルメンテナンス時間を扱い、任意のキー分布の状態において任意のキー入力を与えた時、従来の拡張ハッシュにおいて応答時間を一定値で抑えることが不可能であったが、本論文で提案した改良方式によって応答時間を一定値以下で抑えることができ、ユーザの移動先で個人情報の管理が可能となった。今回はデータ数に比して一

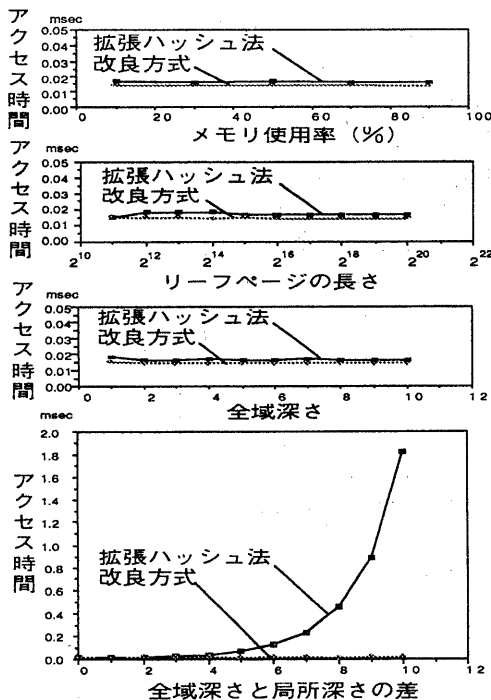


図7. 改良方式のアクセス時間 (下限)

人当りのデータ量が小さい場合についての検討であったが、本方式がAD-DBで用いられることからデータを取り出すまでの時間及び一人当りの記録すべきデータ量の考察も行なわなくてはならない。さらに今後の課題としてAD-DBの配置方法、データの管理方法及びその出力結果を用いた接続制御方法の検討が残されている。

謝辞

本研究を進めるにあたって有益御助言並びに御指導いただきましたNTT通信網総合研究所木下研作グループリーダー、NTT情報通信網研究所中村仁之介主幹員並びにNTT交換システム研究所石川和範主幹員に感謝の意を表します。

参考文献

- 1) 秋山 他, "個人番号サービスにおけるデータベース配置と信号量", 電子情報通信学会 SSE89-22, P.25-30
- 2) M.Fujioka 他, "Hierarchical and Distributed Information Handling for UPT", IEEE Network Magazine, 1990 Nov., P.50-60

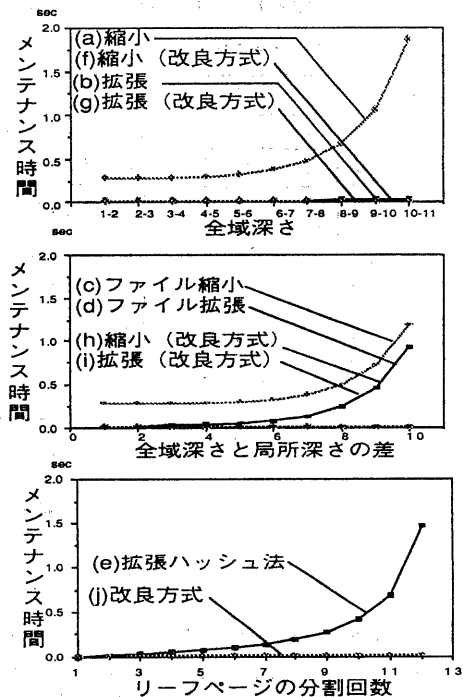


図8. 改良方式のメンテナンス時間

- 3) 水沢, "インテリジェント・ネットワークの周辺", コンピュータ&ネットワークLAN(1990/10, P.28-33, 1990/11, P.8-13)
- 4) 服部, "インテリジェントネットワークにおける知的情報通信", 情報処理学会 DPS-52(1991), P.1-8
- 5) 重田 他, "通信システムにおける分散データ配置方式", 情報処理学会 DPS-45(1990), P.93-100
- 6) 浦, "データ構造", 共立出版(1974)
- 7) 西原, "ハッシングの技法と応用", 情報処理, Vol.21 No.9 (1980), P.980-991
- 8) 弓場, "ハッシングによる見出し探索の技法", 電子通信学会誌, Vol.63 No.1 (1980), P.15-258
- 9) Knuth, "The Art of Computer Programming", Vol.3/ Sorting and Searching, Addison-Wesley (1973), P.506-549
- 10) R.J.Enbody, H.C.Du, "Dynamic Hashing Schemes", ACM Computing Surveys, Vol.20, No.2 (1988), P.85-113 : 遠山訳, 動的hash法, 別冊bit"コンピュータサイエンス", P.43-68
- 11) R.Fagin 他, "Extendible Hashing", ACM TO-DS, Vol.4 No.3 (1979), P.315-344