木構造ハッシュによる分散処理(データ処理アルゴリズム)

安田, 匡祐 / MIURA, Takao / YASUDA, Kyosuke / 三浦, 孝夫

(出版者 / Publisher)
一般社団法人情報処理学会
(雑誌名 / Journal or Publication Title)
情報処理学会研究報告. データベースシステム (DBS)
(号 / Number)
59
(開始ページ / Start Page)
91
(終了ページ / End Page)
98
(発行年 / Year)
2006-05-30

社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－DBS－139 (12)
2006－ＦＩ－ 83 (12)
2006／5／31

木構造ハッシュによる分散処理

安田 匡祐† 三浦 孝夫†

† 法政大学 工学部 情報電気電子工学科 〒184-8584 東京都小金井市梶野町 3-7-2
E-mail: †{c01d3154,miurat}@k.hosei.ac.jp

あらまし 動的ハッシュ技法では，データ量に応じて空間サイズを動的に変化させ効率よい空間量を達成することができる．この代表的技法が線形ハッシュ技法 (LinearHash) である．しかし現実には，一括挿入操作に対して必ずしも効率よい結果を生むとは限らない．本研究では，高度な動的ハッシュ構造である木構造ハッシュ技法 (Tree Hash) を提案する．この技法は複数サーバによる分散処理を想定しており，データ量に対する拡張性を有する．これまで分散動的ハッシュ技法には LH* が知られている．本報告では，実験により LH* と提案手法を比較し，効率よい性能やメッセージ数が達成できることを示す．

Distributed Processes on Tree Hash

Kyosuke YASUDA† and Takao MIURA†

† Dept.of Elect.& Elect. Engr., HOSEI University 3-7-2, KajinoCho, Koganei, Tokyo, 184-8584 Japan
E-mail: †{c01d3154,miurat}@k.hosei.ac.jp

Abstract *Dynamic Hash* allows us to adjust the size of hash space dynamically according to the amount of data so that we obtain the nice time/space efficiency of the hash space. One of the the proposed techniques is *Linear Hash* (LH). However, practically the technique doesn't always provide us with suitable results, especially in the case of batch (consecutive) inserts. In this investigation, we propose a new novel approach, called *Tree Hash* (TH), for the purpose of sophisticated dynamic hash processing. Here we assume distributed environment, i.e., many CPUs and huge amount of data storage connected through high speed network with each other. *LH** has been known as a competitor, and we show empirically the several excellent properties compared to LH*.

1. 前 書 き

インターネットの爆発的拡大により大量のデータを効率よく扱う必要性が増加している．拡張性 (スケーラビリティ) の高いデータ管理・操作の技術は，検索・更新の高い効率を保持することを前提とするものであり，古くて新しい問題の一つである．オンライン実時間環境を前提とする限り，検索更新の計算量を $O(1)$ で実行するハッシュ技法はこの基盤を与えるが，多くの問題点が残されている [1]．すなわちあふれ (衝突) とロードファクタの選定およびハッシュ関数の選択が容易ではなく，ハッシュ(格納) 空間サイズを固定する必要から記憶域利用効率が悪く，部分キー・キー順探索が行えず，あふれの連鎖 (スピルアウト，将棋倒し) が生じる．

動的ハッシュ(Dynamic Hash) は，記憶域効率の改善を目的として提案された．すなわち，ハッシュ(格納) 空間サイズを動的に変更させ，ロードファクタを一定に保つことができる．線形ハッシュ(Linear Hash, LH) はこの代表的手法である．線形ハッシュ技法の特徴は，緩やかな空間の拡張に対応しロードファクタを安定的に保ちながらあふれの解消を行うことにある．確率的に極めて高い性能 (少ない入出力) が期待でき，また実験結果からもこの特性が報告されている．

反面，LH では一括挿入に対する考慮が無く，挿入の都

度バケット分割が発生する可能性がある．実際，バケット分割を発生させる明示的条件は存在しないが，ロードファクタで判断されることが多い．従って，分割の結果がロードファクタを大きく低下させるもので無い限り，分割が連続的に発生する可能性がある．より深刻な問題は，分割が性能向上に結びつかないことにある．空間が線形に拡張されるという本来の性質から，あふれバケットを分割することが困難であり，あふれを解消するための直接的な解決を与えない．このため，時として大きな性能劣化を生むことがある．

分散線形ハッシュ (distributed Linear Hash, LH*) は，分散環境を前提とした大容量ハッシュ空間のためのハッシュ技法であるが，基本的に LH の特性を継承しており何らかの工夫が必要になる．

本研究では分散環境を用いた木構造ハッシュ(Tree Hash, TH) を提案する．この技法では，線形ハッシュと同様にバケットあふれを有し，滑らかに拡張するハッシュである．しかも，上述 LH で問題となるような，一括挿入によるバケット分割の連続発生を抑え，あふれたバケットを分割させあふれ状況を均一化することができる．この結果，検索・更新の性能が大きく向上し，システムの安定性の向上，分散環境によるサーバ負担の軽減が達成できる．

第 2 章で線形ハッシュ手法と LH*手法を要約し，第 3 章で木構造ハッシュ手法を提案する．第 4 章で実験考察により提案手法の優位性を示す．第 5 章は結びである．

2. 線形ハッシュと分散線形ハッシュ

本章では，線形ハッシュ技法 (LH) 及び分散線形ハッシュ技法 (LH*) を要約する．詳細は[2],[3] を参照されたい．

2.1 線形ハッシュ技法

以下では，ハッシュ関数 h が与えられており，与えられたデータ d とキー値 C に対して $h(C)$ によって対応するバケットを指定するとする．バケット集合はハッシュ空間 H を構成し，各バケットは直接アクセスできると仮定される．

線形ハッシュ関数 h_i は更に非負整数レベル値 i をパラメタとして持ち，$h_{i+1}(C)$ は $h_i(C)$ と下 i ビットで同じ値を有すると仮定する．このような関数の例として $h_i(C) = C \bmod 2^i$ が考えられる．更に線形ハッシュでは以下に述べるような"バケット成長値" n (非負整数) を有する．$i=3$，$n=7$ としたときのバケットを図 1(a) に示す．

線形ハッシュ構造における挿入操作は，次のアルゴリズムで実施される．

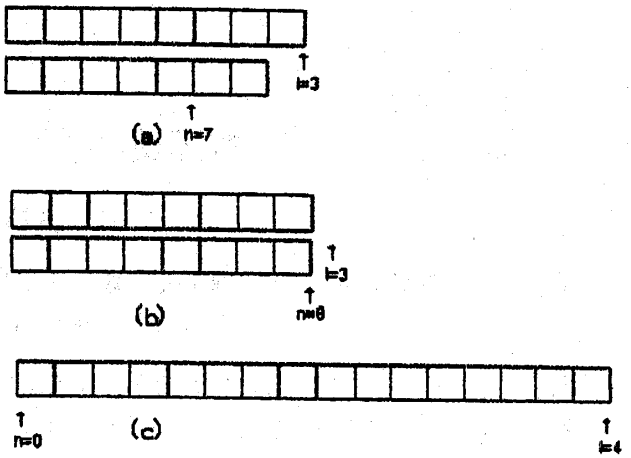

図 1 線形ハッシュ

$a \leftarrow h_i(C);$

$if\ a < n\ then\ a \leftarrow h_{i+1}(C);(A1)$

即ち，$h_i(C)$ がバケット成長値を下回れば，レベルを $i+1$ として再計算し，$i+1$ ビット長の値として見る．格納場所が確定すれば，必要ならあふれを許して当該バケットに保存される．

挿入の結果，ロードファクタ (バケットデータ総数に対する実際の格納データ数の割合) がしきい値を超えたとき，バケット分割が発生する．分割は，バケット成長値 n で示されているバケットが選ばれ，当該バケットを $i+1$ レベルのハッシュ関数 (この値は $h_i(C)$, $h_i(C)+2^i$ のいずれか) に応じて再分配する．ハッシュ空間サイズが 1 だけ増加したことに注意したい．

バケット成長値は，次のアルゴリズムにより再計算される．

$n \leftarrow n+1;$

$if\ n \geq 2^i\ then$

$n \leftarrow 0$

$i \leftarrow i+1;$

この様子を図 1(b) に示す．成長値がバケットの数 2^i+1 に達するとレベル i を増加し，n を 0 に戻すことを示している (図 1c)．

すでに示したように，LH が現実に有する問題が上記のアルゴリズムに由来する．実際，ロードファクタの低下が少ないため，一括挿入により連続したバケット分割が発生する．また，あふれバケットが分割するとは限らず，バケットに偏りが生じる．

2.2 分散線形ハッシュ

LH* が仮定する分散環境では，高速ネットワークを用いてサーバ同士のメモリにアクセスし，総体的に大容量となる環境を仮定する．LH* では線形ハッシュに基づいたデータ管理を行う．各サーバは独自にバケット空間を有し，またバケットレベル i，バケット成長値 n を有する．LH* ではクライアント計算機を仮定する．

各クライアントは各サーバ機ごとの i', n' を管理する．この値は，サーバのそれと異なるものであってもよい．データの挿入に対し，クライアントは挿入キー C と i'，n' を用いてアルゴリズム (A1) による「クライアントアドレス計算」(A1') を行い，その結果をサーバアドレスとみなして，該当サーバにデータとキーを送信する．

$$a \leftarrow h_{i'}(C);$$
$$if\ a < n'\ then\ a \leftarrow h_{i'+1}(C); (A1')$$

(A1') によって求めた値 a が正しくない可能性があるため，受け取ったサーバは受信メッセージが正しい相手に送られているのかを確かめるために「サーバアドレス計算」(A2) を行う．

$$a' \leftarrow h_j(C);$$
$$if\ a \neq a'\ then$$
$$a'' \leftarrow h_{j-1}(C)$$
$$if\ (a'' > a\ and\ a'' < a')\ then\ a' \leftarrow a''; (A2)$$

計算値 a' が a と一致するときは正しいが，そうではない場合には，"本当のサーバ" a' に要求を転送し，サーバ a' はこの手順を繰り返して正しいサーバを計算する．

この操作は高々 2 回の転送で完了することが知られており，これにより挿入時に最低 1 回，最高 3 回のメッセージ転送が行われることになる．検索時は検索結果が返送されるため，最低 2 回，最高 4 回のメッセージ転送が必要となる．

転送が発生した場合，クライアントの i', n' が実際のものと違っている．このとき，正しいサーバから正しいバケットレベル i が送られ，クライアントは (A3) を用いて実際の値により近い i', n' に更新する．

$$if\ j > i'\ then$$
$$i' \leftarrow j - 1$$
$$n' \leftarrow a' + 1;$$
$$if\ n' > 2^{i'}\ then$$
$$n' \leftarrow 0$$

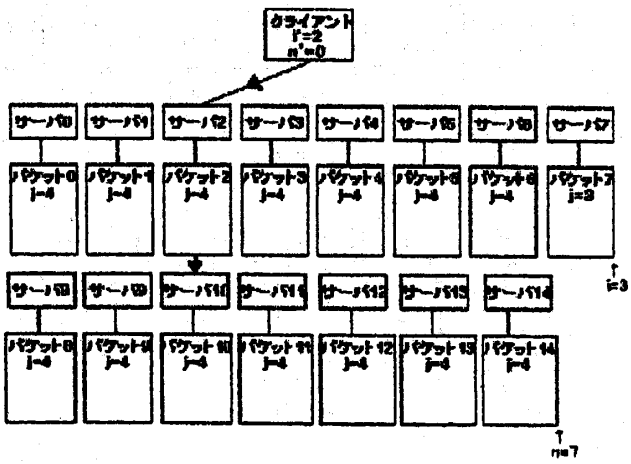

図 2 LH*の挿入

$$i' \leftarrow i' + 1; (A3)$$

[例 1] キー C を用いた挿入を図 2 に示す．$i = 3$，$n = 7$，キー $C = 26$ で挿入を行う．クライアントの i'，n' は $i' = 2$，$n' = 0$ とする．(A1') を用いた結果 $a' = 2$ となり，クライアントはサーバ 2 に挿入メッセージを送る．メッセージを受け取ったサーバ 2 の i' は 4 であり，(A2) を用いた結果 $a' = 10$，$a'' = 2$ となる．$2 \neq a'$ を満たすためメッセージはサーバ 10 に転送される．サーバ 10 が (A2) を用いた結果 $a' = 10$ となり $a = a'$ が満たされデータはバケット 10 に挿入される．

挿入によってバケットがあふれたとき，バケット分割が発生する．このときサーバは分割調整サーバにメッセージを送り，分割調整サーバは所持している n で示されるサーバに分割のメッセージを転送し，以下を計算する．

$$n \leftarrow n + 1;$$
$$if\ n \geqq 2^i\ then$$
$$n \leftarrow 0$$
$$i \leftarrow i + 1;$$

メッセージを受け取ったサーバ n はバケットレベル $j+1$ のバケット $n + 2^j$ を作成し，h_{j+1} でハッシュしてバケットを分割・再配分しバケットレベルを更新したあと，分割調整サーバに分割完了を報告する．分割においてメッセージ転送は 4 回行われる．また分割調節サーバは常に正しい i，n を有することに注意したい．

この結果，メッセージ転送回数は挿入では 1，最悪で

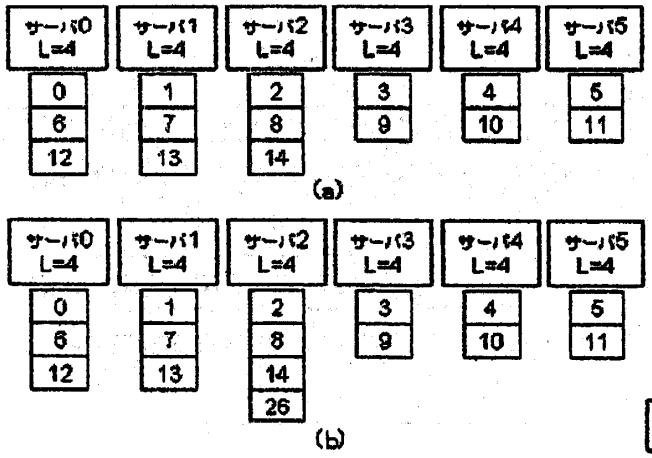

図 3　木構造ハッシュのバケット

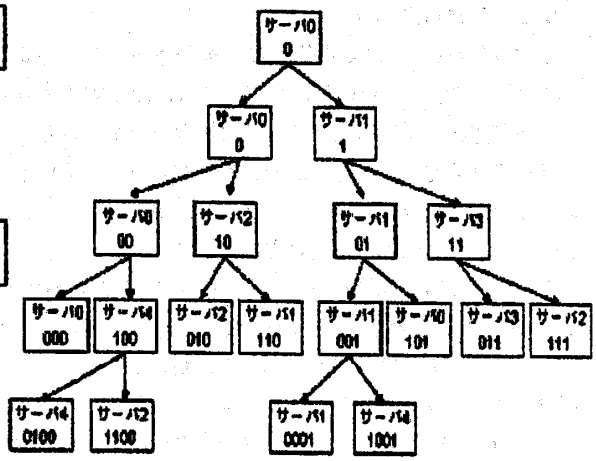

図 4　木構造ハッシュのイメージ

も 3，検索は 2，最悪でも 4 で完了するため，拡張性 (スケーラビリティ) の問題は改善することができる．しかし，LH の有する問題は改善されておらず，分割調整サーバが必要であり，分割すべきバケットの選定が困難である．

3. 木構造ハッシュ

3.1 木構造ハッシュのねらい

分散環境を使用した動的ハッシュである木構造ハッシュ手法について述べる．本稿で提案する木構造ハッシュの狙いは，動的ハッシュの特性と拡張性を保持しつつ，線形ハッシュや分散線形ハッシュで生じるような，バケット分割に伴う問題の改善にある．このため木構造と分散環境を併用し，木構造に見られるバランスの悪化を防ぐためにバケット全体をサーバで保持し，均一にサーバに配置する，サーバごとの偏りを防ぎ，木構造の偏りも問題にはなりにくい．この形を図 3(a) に示す．サーバの数に制限はないため，任意規模の環境で使用できる．

木構造ハッシュは n，i は用いず、高さの情報のみを用いる．高さ i のバケットが分割した時高さは $i+1$ となり，キー c を挿入する場合ハッシュ値は $h_{i+1}(c)$ で求められる．分割されたバケットは上 1 ビットのみ分割前と異なる．1 ビットが 0，1 で表されるとき，バケットは図 4 のよう分割されていく．この形は基点となるバケット 0 を根，分割されるバケットを節，分割されていないバケットを葉とすれば，木構造と見なすことができる．図では葉，節，根と 3 種類のバケットが存在するが，実際に存在するバケットは葉のバケットのみである．

3.2 検索挿入操作

N 個のサーバは非負整数でアドレス可能であり，各サーバはそれ自身のアドレスを知っていると仮定する．各サーバは，データを格納するためのバケット空間と，バケットアドレスを管理するアドレス表を有する．各バケットは非負整数でアドレシングされ，更に各バケットには自らのバケットレベル m が付与されており，サーバはその最大値をサーバレベル L として保持すると仮定する．なお全てのサーバで初期値は $L=0$ である．

クライアントから検索・挿入の要求を受けたサーバは，以下を計算する．

$$a \leftarrow h_L(C) = C \bmod 2^L; (B1)$$

サーバはバケットアドレス表を調べ，当該バケット a を自らが管理すると判断したとき，この検索・検索を行う．この結果，バケット a が存在しないとき，$L' = L$ として以下を計算する．

$$L' \leftarrow L' - 1;$$
$$a \leftarrow h_{L'}(C); (B2)$$

この結果，当該バケット a を自らが管理すると判断する限り，a が見つかるまで (B2) の計算を行う．

当該バケット a を自らが管理しないと判断したとき以下を計算する．

$$a' \leftarrow C \bmod N;$$

この値 a' を用いてサーバはサーバ a' にメッセージを転送する．メッセージを受け取ったサーバ a' は (B1) を用いてアドレスを計算し，これまでの手順を繰り返す．

サーバの検索は葉からさかのぼり，節で目的のバケット

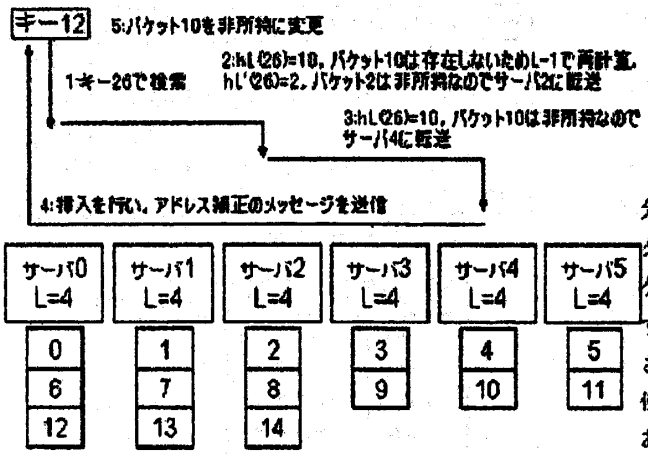

図 5　木構造ハッシュの挿入

へと方向を変える．サーバは転送の都度木を底辺からさかのぼる事になるため，高々$MaxL = min$(全サーバのLのうちで最大の$L, N) - 1$回の転送が行われる．このため，挿入では最低1回，最高$MaxL$回，検索は最低で2回，最高で$MaxL + 1$回のメッセージ転送が必要となる．

3.3　アドレス補正

メッセージが転送され，検索・挿入されるべきサーバは，最終的に正確なバケットアドレスa，バケットレベルm'をクライアントへ送信する．クライアントはこの値を用いてバケットアドレス表を更新する．

$if(m' > L)\ then\ L \leftarrow m';$
$while(m' > 0)($
$a \leftarrow h_{m'}(C);$
$if($アドレス $(a) = NULL)$
アドレス $(a) \leftarrow$ 非所持;
$m' \leftarrow m' - 1;$
$)$

挿入およびアドレス補正の状況を図5に示す．$MaxL$はL及び サーバ数Nに応じて大きくなるが，一度のアドレス補正で複数のアドレスの補正が行えるため，実際の送受信回数は低い値となる．

3.4　バケット分割

THにおけるバケット分割は以下の条件を満たすときに発生する．本稿では以下の4つを分割条件として設定する．

（1）$\frac{\text{サーバ内の全レコード数}}{\text{バケット数}\times\text{バケット容量}}$が一定以上，かつバケットがあふれている

（2）バケットからあふれたレコード数が一定以上

（3）バケットのレベルがLより一定以下で，バケットがあふれている

（4）分割直後のバケットのレコード数が一定以上

条件(1)により，サーバ内のロードファクタをバケット分割のきっかけとできる．これにより全体のロードファクタを一定の値に近づけることができる．また，あふれバケットのみを分割する．条件(2)はあふれの長さを一定にすることにより，あふれ処理を伴う検索での入出力を減少させることができる．条件(3)では，バケット分割直後で使用領域が低下しロードファクタを満たさないときでも，あふれが存在すると思われるバケットを分割することができる．条件(4)により，分割された直後のバケットあふれを回避できる．

バケットあふれだけで分割を行う場合，ロードファクタの低下を伴うので，記憶域効率が下がる(注1)．ただ，バケット分割が頻繁に発生し，入出力回数が増大しがちとなる．バケット毎のあふれデータ数を管理しない場合，条件(2)による分割は検索時にのみ行われる．

バケット分割が発生すると，当該バケットaは自身のバケットレベルmを用いて以下を計算する．

$a' \leftarrow a + 2^m;$
$m = m + 1;$
$a'' \leftarrow a' \bmod N;$
$if\ (m > L)\ then\ L \leftarrow m;$

この結果，バケットをmを用いてハッシュしてaとa'に分割し，バケットa'及びバケットレベルmを，サーバa''に分割メッセージとして送信する．

受信サーバでは次の計算を行い，バケットアドレス表を更新する．

アドレス $(h_m(C)) \leftarrow a'$のアドレス;
$m' \leftarrow m - 1;$
$if\ (m > L)\ then\ L \leftarrow m;$
$while(m' > -1)($
$a \leftarrow h_{m'}(C);$
$if\ ($アドレス $(a) = NULL)$
アドレス $(a) \leftarrow$ 非所持;
$m' \leftarrow m' - 1;$
$)$

(注1)：経験的に 60 パーセント程度になる．

なお，各サーバ毎にバケット分割の判断を行い，あふれバケットが分割することがあっても独自で判断するため，LH* のような分割調整サーバは必要がない．またメッセージ転送も 1 回ですむ．

[例 2] 図 5 の挿入でバケット分割条件が満たされた時バケット分割が発生する．バケット 10 の m は 4 であり，バケット 10 とバケット 26 に分割される．$26 \bmod 6 = 2$ より，バケット 26 はサーバ 2 に送信される．バケット 10，バケット 26 の m は 5 となり，$m > L$ が満たされサーバ 2，サーバ 4 の L は 5 になる．バケット分割によりバケットは図 3(b) になる．

4. 実　　験

4.1 準　　備

本章では木構造ハッシュの有効性を検証するため，様々な実験を通じて LH, LH* との比較検証を行う．

本稿での実験環境として使用するデータ，評価対象の値，実験に使用するバケット分割条件，実験内容の詳細について述べる．挿入，検索に使用するデータとして 120,000 件の郵便番号を用いる．Java 言語を用いてマルチスレッドによる分散環境をシュミレートする．メッセージ転送は，通信環境のみを想定し共有メモリを仮定しない Shared-Nothing 方式を用いる．この結果，スレッドごとのコミュニケートはメッセージのやり取りによる．

評価の対象は使用したデータあたりの平均入出力 (I/O) 回数，平均メッセージ送信回数とする．I/O はバケットに読み書きをしたときにカウントされ，バケットアドレス表はメモリに存在するとする．I/O 回数の計測は，線形ハッシュに対しても行い，メッセージ数計測は LH* との比較に利用する．LH* との比較は文献 [2] の結果を使用する．

本実験では，バケット分割条件として以下を用いる．

（1） 閾値 90 パーセント

（2） バケットレベルが $L-2$ でバケット分割

（3） あふれが 30 以上でバケット分割

（4） あふれがバケット容量の 30 パーセント以上でバケット分割

（5） 分割後のバケットの使用量が 80 パーセント以上で分割

上述のように，あふれの長さを用いた分割は検索時にのみ行われることに注意したい．

以下ではいくつかの項目について実験を行う．

4.2 挿入件数

まず始めに挿入件数を変更しての実験を行う．サーバ数は 3，バケット容量は 50，データ件数を 1000, 5000,10000,50000,100000 件のそれぞれで 連続挿入を行い，その後で挿入したデータを検索する結果を表 1，表 2 に示す．

挿入件数	LH 挿入	LH 検索	TH 挿入	TH 検索
1000	4.885	2.077	4.739	1.202
5000	5.024	3.018	4.932	1.006
10000	5.053	2.990	4.792	1.014
50000	5.098	1.211	4.243	1.132
100000	5.102	1.197	4.187	1.138

表 1　挿入件数・I/O 回数

挿入件数	挿入	検索	LoadFactor
1000	1.044	2.014	68
5000	1.042	2.000	75
10000	1.043	2.000	75
50000	1.036	2.000	86
100000	1.036	2.000	89

表 2　挿入件数・メッセージ回数

バケット容量	LH 挿入	LH 検索	TH 挿入	TH 検索
10	5.424	1.488	5.292	1.000
20	5.206	1.612	4.955	1.002
50	5.099	1.212	4.243	1.132
100	5.035	1.207	4.323	1.127

表 3　バケット容量・I/O 回数

LH における挿入は，挿入件数が多くなると I/O 回数が増大し，検索は挿入件数が多いと I/O 回数は 1.2-1.9 回程度だが，少ないと 2.1-3.0 回と多くなる．一方 TH の場合は何れも LH よりも良い結果となっている．特に検索においては何れの I/O 回数も 1.0-1.2 回の間に収まっている．

メッセージ回数について，挿入では 1.04 回，検索では 2.00 回と理想に近い値になっている．転送回数は 50000 件のときで 354 回と多いが，1097 回生じるバケット分割による生じるメッセージ数が少ないため，メッセージ回数を増やさない理由となっている．

TH のロードファクタは，挿入件数が多くなると設定した値である 90 パーセントに近づく．これはバケットに偏りが生じ，1 回だけ多く分割されるバケットが生じたときに「バケットレベルが一定以下で分割」条件が発生しやすいからである．

4.3 バケット容量

2 つ目の実験はバケット容量を変更して行う．データ件数を 50000 件，サーバ数は 3，バケットの容量を 10,20,50,100,200 のそれぞれで連続挿入を行い，その後で挿入したデータを検索する．結果を表 3，表 4 に示す．

LH での挿入は，バケット容量が大きくなるにつれ I/O 回数が減少し，検索では常に 1.3-1.6 回程度の I/O となっている．一方 TH の場合は何れも LH よりも良い結果となっている．特に検索においては何れの I/O 回数も 1.00-1.13 回の間に収まっている．LT も TH もバケット容量が 50 以上になると 1.2 以下の I/O 回数になる．

バケット容量	挿入	検索	LoadFactor
10	1.269	2.003	68
20	1.136	2.001	67
50	1.036	2.000	86
100	1.018	2.000	89

表 4　バケット容量・メッセージ回数

サーバ数	挿入	検索
1	4.135	1.050
3	4.243	1.132
5	4.303	1.137
10	4.434	1.116
20	4.519	1.140

表 5　サーバ数・I/O 回数

メッセージ回数について，挿入では 1.00-1.26 回，検索では 2.00 回と理想に近い値になっている．これもバケット容量が大きくなることによりバケット数が少なくなりバケット分割時のメッセージ回数が減少するためである．

バケット容量が少ないときロードファクタの値は 67 これはバケット容量が少ないため，連続でバケット分割が発生するからである．

4.4　サーバ数

3 番目の実験はサーバ数を変更して行う．データ件数を 50000 件，バケットの容量は 50，サーバ数を 1,3,5,10,20 のそれぞれで連続挿入を行い，その後で挿入したデータを検索する．結果を表 5，表 7，サーバ毎の I/O 比率を表 6 に示す．

サーバ数	挿入	検索	倍率 挿入	倍率 検索
1	4.135	1.050	1.000	1.000
3	1.468	0.377	2.816	2.785
5	0.856	0.229	4.828	4.577
10	0.437	0.120	9.452	8.755
20	0.229	0.062	18.073	16.915

表 6　サーバ数・倍率

サーバ数	挿入	検索	LoadFactor
1	0.000	0.000	75
3	1.036	2.000	86
5	1.047	2.000	89
10	1.059	2.001	89
20	1.066	2.001	89

表 7　サーバ数・メッセージ回数

検索時サーバが 1 つの時は 1.05 回と最良だが、それ以外では 1.14 回程度となっている．これはコミュニケーションにおいてオーバヘッドが発生しているのが原因だが，少ない値だといっても良い．

バケット容量	LH* 挿入	LH* 検索	TH 挿入	TH 検索
17	1.437	2.008	1.190	2.017
33	1.231	2.007	1.085	2.000
64	1.120	2.007	1.028	2.002
125	1.064	2.006	1.015	2.001
250	1.033	2.006	1.007	2.002
1000	1.009	2.004	1.002	2.000

表 8　LH*との比較・メッセージ回数

サーバ数が多くなるにつれ，送信のメッセージ転送回数が増加するが，サーバ数が倍になってもメッセージ送受信回数の増加はおおよそ 0.01 回と非常に低い値となっている．

4.5　LH*との比較

4 番目の実験は LH*との比較を行う．データ件数を 10000 件，サーバ数は 3，バケットの容量を 17,33,64,125,250,1000 のそれぞれで連続挿入を行い，その後で挿入したデータを検索する．結果を表 8 に示す．

検索においてバケット容量が 17 の時を除き，TH は LH*と比べ 0.10-0.24 回ほどメッセージ回数が少ない．バケット容量が 17 の時は検索時にバケット分割が発生したのが増加の原因である．LH* はバケット容量が 17 のときバケットの数は 1,012 となり，1011 回もバケット分割が生じ，バケット分割だけで 4044 回のメッセージ送受信が行われることになる．LH* の再送回数は 161 回だが，TH の再送回数は 300 回と倍近くある．よりメッセージを必要とするバケット分割に必要なメッセージは 1/4 程度なので，結果的に必要なメッセージは少なくなっている．

4.6　バケット分割条件ごとの比較

5 番目の実験はバケット分割の条件を変更して行う．データ件数を 50000 件，サーバ数は 3，バケットの容量を 20,50,100 のそれぞれで次の条件を使用し連続挿入を行い，その後で挿入したデータを検索する．

（1）　ロードファクタでの分割

（2）　(1) 及びバケットレベルが L -2 で分割

（3）　(1) 及び分割後レコード数がバケットの容量の 80 パーセント以上のときに分割

（4）　(1) で挿入し，検索時あふれが一定以上のとき分割

（5）　(1)-(4) を全て使用

以上の条件を TH(1),..,TH(5) とする．結果を表 9 に示す．

挿入では，TH(3) 以外は，LH よりも I/O 回数は少ない．何れもバケットの容量が高くなると I/O 回数が減るが，特にバケットレベルが一定以下で分割する場合は低い I/O 回数で安定している．しかし検索では一転し，バケットレベルを用いる TH(2) と，全てを用いる TH(5) のみが良い結果となった．ロードファクタのみを用いた場合は，

容量	処理	LH	TH(1)	TH(2)	TH(3)	TH(4)	TH(5)
20	挿入	5.206	5.050	5.065	4.876	5.050	4.955
50	挿入	5.099	4.817	4.669	5.140	4.817	4.243
100	挿入	5.035	4.723	4.681	4.750	4.723	4.323
20	検索	1.612	1.454	1.147	2.727	1.253	1.002
50	検索	1.212	1.827	1.156	1.822	1.396	1.132
100	検索	1.207	1.926	1.123	4.343	1.162	1.127

表 9 バケット分割条件ごとの比較・I/O 回数

多くあふれているバケットに挿入が行われたとしても，分割が起こるとは限らない．この場合，バケットのレベルは低い場合が多く，バケットレベルでの分割を行うことによりあふれを均一化できる．あふれバケットの長さでの分割は検索時に行われるため，TH(4) と TH(5) はこの値はバケット分割に必要な I/O 回数も含まれている．連続で同じデータを検索した場合 I/O は 1.07-1.10 回程度まで減少する．

4.7 あふれバケットの長さ

6 番目の実験はあふれの長さを変更して行う．サーバ数は 3，バケットの容量は 50，データ件数を 10000,50000 件のそれぞれで連続挿入を行い，その後で挿入したデータを含む 100000 件のデータで検索する．結果を表 10 に示す．

挿入件数	LH	TH(1)	TH(2)	TH(3)	TH(4)	TH(5)
10000	10.118	6.737	3.344	9.549	1.554	1.023
50000	2.176	3.604	1.901	5.708	1.706	1.682

表 10 あふれバケットの長さ・I/O 回数

バケットにレコードが存在しない場合，あふれデータが最後まで読み込まれるので I/O 回数が増大する．TH(2), TH(4) では線形ハッシュよりも I/O 回数は少なく，あふれが均一化される．とくにあふれバケットの長さを用いた場合は，一定以上のあふれで分割されるため，あふれは均一となる．

4.8 考　　察

実験結果から，ほとんどの場合で TH が LH よりも効率よい結果を示している．これらの結果より I/O 回数，利用効率，メッセージ回数の 3 項目を考察する．

TH の挿入時の I/O 回数が減る条件として (1) データ件数が多い (2) バケット容量が大きい (3) 使用するサーバ数が少ない (4) サーバレベルを用いた分割を行う，が挙げられる．(1)，(2) が満たされたとき検索の I/O 回数は増加するが，同時に安定する．サーバを複数使った場合全体の I/O 回数は増加するが，そのオーバヘッドは少ない．

木構造ハッシュのロードファクタが上昇する条件としては，(1) バケット容量が大きい (2) 使用するサーバ数が多い (3) 分割後の連続分割を行わない，が挙げられる．(1) は I/O 回数が減少する条件と同一である．バケット容量を多くすることにより，I/O 回数を減らし、利用効率を改善する．(2) の条件より 1 サーバで TH を用いる場合，高いバケットを用いなければ利用効率は悪化する．

TH メッセージ送受信回数が減少する条件として (1) バケット容量が大きい (2) 使用するサーバ数が少ない，が挙げられる．バケット容量が減少すると，メッセージ送受信回数は増加する．これはアドレスエラーが頻繁に発生するからである．それに対しサーバ数を増加させる場合，メッセージ送受信回数の増加は少なく，分散環境に適している．また挿入，検索の転送回数は最大で min(全サーバの L のうちで最大の L, N) -1 回だが，サーバ数に関係なく 1 件ごとのメッセージ送受信回数は挿入でおおよそ 1 回，検索で 2 回と理想に近い形になる．

5. 結　　び

本稿で示した実験により，木構造ハッシュ(TH) は，線形ハッシュ(LH)，LH* の持つ問題を解決し，両手法よりも高い効率を有する動的ハッシュ構造を提供することがわかる．特に，バケット分割（合併）条件を設定することにより，TH はあふれバケットを的確に分割することができる．これにより偏りの少ない効率よい個かを期待することができる．TH は分散環境を想定しているが，1 サーバであっても効率が良い．分散環境では I/O 負荷を簡単に軽減することができるため，導入・移行が容易である．また分割調整サーバを用いないことに着目したい．

いくつかの問題点も存在する．まず，ロードファクタ条件が比較的不安定である．今回の実験では分割条件すべてを使用したが，一部条件だけではロードファクタは安定するものの I/O 回数は増大してしまう．また，TH 構造から削除を行いバケット合併が必要となっばあい，合併バケットの候補の選定が，その後の処理効率に影響することが予想できる．

謝　　辞

本研究の一部は文部科学省科学研究費補助金 (課題番号 16500070) の支援をいただいた．

文　　献

[1] シェーファー, C.: データ構造とアルゴリズム解析入門, ピアソンエディケーション, (原著 1998)

[2] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider: "LH* - Linear Hashing for Distributed Files", In *SIGMOD Conference*, pp. 327-336, 1993.

[3] Witold Litwin: "LINEAR HASHING : A NEW TOOL FOR FILE AND TABLE ADDRESSING", In *Proc. of VLDB*, 1980.