メモリＤＢＭＳにおけるハッシュ関数の最適化*

6Ｆ－5

日高東潮　板倉一郎　黒岩淳一†
ＮＴＴ情報通信研究所‡

1　はじめに

近年メモリの低価格化により、データ参照時の高速なレスポンスをディスクI/Oの回避により実現する、主メモリ常駐DB(以下メモリDB)などの、実メモリ上データ配置方式のDBMSが普及しつつある。

そのメモリDBにおいて、ディスクベースDBMSでは得ることのできない、主記憶上のデータに対するランダムアクセスの高速性に注目したインデックスの一つとして、ハッシュインデックスが挙げられる。

本稿では、メモリDBにハッシュインデックスの採用を考慮する場合、データ投入された状況下での、データアクセスの高速性とメモリ使用量の両面を考慮した、ハッシュ関数の決定方法について考察する。

2　メモリDBにおけるハッシュ関数の問題点

現在研究中のメモリＤＢ [1] では、インデックスにStatic-Hashingの高速アクセス性とDynamic-Hashingの拡張性の両特性を合わせ持つハッシュ木を採用している。(図１)

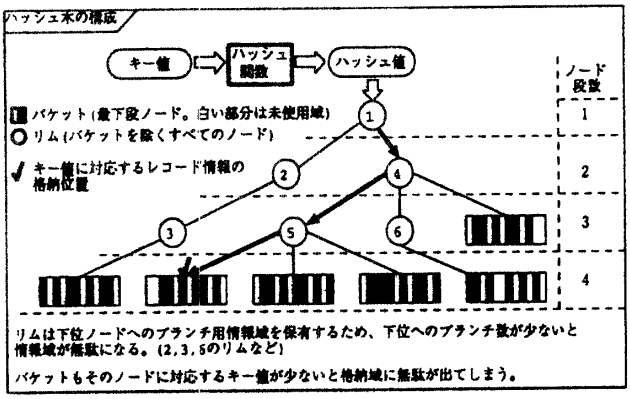

図１：ハッシュ木におけるメモリ使用状態のイメージ

しかし、本インデックスにおいて運用上、内部で使用するハッシュ関数に以下のような問題がある。

- 少ない演算量で十分なキー値分散を得られるハッシュ関数の構築が困難

  ディスクI/Oが無いため、データ検索処理にハッシュ関数の計算が占める割合は比較的大きくなる傾向にあるが、必要なキー値分散を得るためには複雑な演算となりやすい。現在のところ、ハッシュ演算量はキー長依存ではあるが、キー値が与えられ、ハッシュ値をもとにレコード参照を行なう行程に占めるハッシュ関数演算量は約１割程度であるが、今後複雑なサービスを実現することを考えると演算量はさらに増加すると考えられる。

- ハッシュ関数によりメモリ使用率が変化する。

  キー値分散の不足でインデックスの空き領域が多数出来てしまい、容量の限られたメモリDBでの運用を困難にしてしまう危険性がある。

以上の問題を回避するため、複数のハッシュ関数をシステム側で用意しておき、ユーザーが必要に応じて関数を選択する方式が多く見られるが、投入されるデータにより、ハッシュ値は必ずしもデータ格納空間に均一に分散する保証が無いため検証が必要となる。またデータベースのサービス当初は十分に分散していたデータも、データの更新が行なわれるに従いデータの格納空間内における配置の分散が十分で無くなるケースが多々あり、高速なデータアクセス性能を安定して引き出す上でハッシュ関数の選択が、ＤＢＭＳユーザーに運用上の負担を強いている要因となっている。

3　ハッシュ関数の最適化

従来議論されてきたハッシュ関数としては除算法、平方採中法、基数変換、折り畳み法などが挙げられる。[2] ハッシュ関数の最適化を行なうことを考えた場合、上記の手法を用いることを前提とすれば、各手法で用いるパラメタの最適化が考えられる。

しかし、これらの手法は一般的なキー値に対する手法であるため、業務の性質により、キー値になんらかの規則が存在する場合、上記の定数パラメタのみ可変で演算方法が固定の方法より、演算方法も可変な関数構成を前提に関数最適化を行なう方が有効なことが多い。

ここでは、ハッシュ関数を生成し、それに対する評価関数を与え、ハッシュ関数から近似最適解として有用なハッシュ関数を絞り込む枠組として図２の構成を考え、次節より図２を構成するハッシュ関数空間の生成方法、ハッシュ関数の評価関数、最適化アルゴリズムのそれぞれについて考察する。

*An Optimization of Hash function on Main Memory Resident Database Systems.

†Toshio Hitaka, Ichiro Itakura, Junichi Kuroiwa
‡NTT Information and Communication Systems Laboratories 1-1 Hikarinooka,Yokosuka,Kanagawa 239 Japan

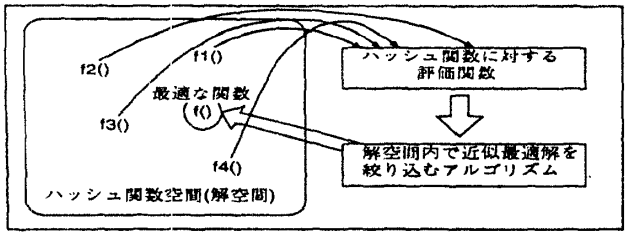

図2：ハッシュ関数最適化の枠組

3.1 最適化するハッシュ関数の構成

本稿では、さまざまな演算構成のバリエーションを持つ関数を生成するにあたり、以下の前提の下に考えるものとする。

1. 検索処理に占めるハッシュ関数の演算コストを考慮し、一定以上の演算を行なわない。
2. 演算に使用するパラメタはCPU内部の処理を考慮し、キー値をワード単位で分解したものと、数個の定数を用い、ハッシュ値の返却はワード単位で行なう。

ハッシュ関数の引数のイメージは図3のようになる。

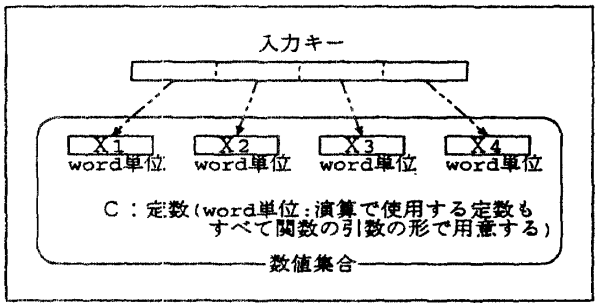

図3：ハッシュ関数の引数：X_i

ハッシュ関数の演算構成は演算量の簡易な制御を可能にするため、複数ワードで構成されるキー値／定数と、CPUの1命令にて演算可能な原始演算である4則演算／シフト演算／論理演算を組み合わせて構成する方式を考える。(図4)

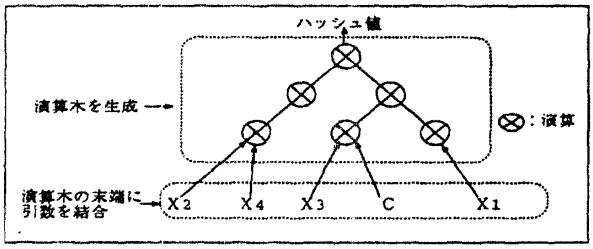

図4：ハッシュ関数のイメージ

演算の組合せは演算量を抑制するため、演算子数の上限を固定し、変数の複数回使用を許容し構成する。

3.2 最適化における評価関数 $F(f)$

ハッシュ関数：$f(X_i)$ に対する評価は以下の観点で行なう。

- 評価関数：$F(f)$ の最大化問題とする。
- メモリ使用量：$M(f)$ が初期の規定値：M_0 を越えたら評価関数：$F(f) = 0$。
- ハッシュの分散が小さく、ハッシュ木内格納段数の分散：$D(f)$ が大きい場合分散の度合に従い評価関数値に負の値$-D(f)$ を加える。
- 演算量：$Cal(f)$ の少なさに応じて評価関数値に正の値：$Cal(f)$ を加える。

以上より、評価関数：$F(f)$ は以下のようになる。

$$F(f) = \begin{cases} -D(f) + Cal(f) & M(f) \leq M_0 \\ 0 & M(f) > M_0 \end{cases}$$

3.3 最適化アルゴリズムに対する考察

最適化アルゴリズムとしては、さまざまな手法が存在するが、今回の最適化においては「解空間内の特性が不明」という前提があるため、解空間に対する評価関数の連続性が重要な近似最適化やSimulated-Aniering（焼きなまし法）などの手法ではなく、遺伝的アルゴリズム[3]が有効であると考える。

遺伝的アルゴリズムの特徴としては、2つの近似最適解を元にした、それぞれの特徴を部分的に引き継ぐ解も近似最適解になりやすいという前提条件の下、解空間における連続性や、解空間における厳密解では無い近似最適解の影響を受け難いこと、並びに時系列における解空間の変動に強いことなどが挙げられる。

また遺伝的アルゴリズムのデメリットとしては演算量が大きいことなどが挙げられるが、本稿ではDBMSの運用中に動的な最適化を行なうことは考慮しておらず、オフラインでの最適化を行なうことのみを目的とし、遺伝的アルゴリズムのデメリットによる問題は無い。

4 最後に

従来経験に基づき決定されていたハッシュ関数を、投入データに基づき自動的に決定する手法を提案した。

今後、実際の運用データに基づき、提案手法により比較的安定した性能を得ることが可能なハッシュ関数が生成できるかを検証し、検証を通じて評価関数で使用するパラメタなどについての考察を行なう予定である。

参考文献

[1]T.Honishi, N.Kobayashi and J.Nakamura: Design and Implementation of an Enhanced Relationa Database Mabagement System for Telecommunication and Network Applications, in Proceedings of Pacific Telecommunications Council Eighteenth Annual Conference, pp.698-703,1996

[2]Michael J. Folk, Bill Zoellick: File Structures, Addison-Wesley Longman, 1992

[3]David E.Goldberg: GENETIC ALGORITHMS in Search, Optimization, and Machine Learning, 1989