⑯ Asymptotic Load Balance of Distributed Hash Tables

N.Ichiyoshi*, K.Kimura(ICOT,日本)

発表要旨

ハッシュテーブルで, bucketをプロセッサ数だけ等分割し，各々をプロセッサに割り当てて並列化したものを分散ハッシュテーブルという。多くのsearch/insert操作を同時に扱うことができるため，プロセッサ数をpとすると，スループットは逐次処理の場合のp倍まで増加する。しかし，平均的なケースでは，負荷のアンバランスにより最大のスループットには到達しない。我々は，平均的なケースで負荷バランスを保つために必要なpに対するq（＝各プロセッサに割り当てるbucketの大きさ）の増加率について研究した。我々が用いた確率モデルでは，load factor（bucket毎の要素数の平均）は一定，bucket 〈i,j〉(i番プロセッサのj番bucket）への要素数はPoisson分布に従うと仮定している。シミュレーションでのinsertオペレーションに対し解析した結果, $q=\omega((\log_2 p)^2)$で十分であることがわかった。本研究で用いた負荷バランス解析は，実行マシンに依存せず，他の問題にも適用可能である。

質疑応答

質問：結論の最後から２番目（ネットワーク通信オーバーヘッド）には賛同する。pを増やす時qを一定に保てば，ネットワーク通信のオーバーヘッドはp^2に増える，そうなると，問題サイズをさらに速く増加しないといけない。疑問に思うのは，問題サイズがそれほど速く大きくならない場合，単にハッシュテーブルのサイズを増やすだけなら，例えば，サイズの増加と同じ速さで増加する数でプロセッサへ分散させるハッシュ関数を用いねばならないと思う。問題サイズを増やす必要がない時の分散はどうなるのか？

回答：（“Average Load Balance of Distributed Hash Tables”のグラフで説明）我々は問題サイズは少なくとも$\log_2 p$程度の速さで増加すべきであると見ている。我々はもし負荷が割り当てた要素数に比例するなら，負荷のバランスを保つためには少なくともqは$\log_2 p$に比例して増えねばならないと仮定している。つまり，必要最小のqの増加率である。もし，$(\log_2 p)^2$があなたのプログラムにとって大きすぎるなら，あなたが述べたテクニックを使わねばならないだろう。それは，二つのハッシュ関数で，一つはプロセッサに要素を分散するための，もう一つはプロセッサ内部のハッシュテーブルの処理のための関数である。qが固定の場合，負荷を増やさないためにバケット分割の手法を使うことになるかもしれない。まだ，検証はしていないが，分析結果は分散ハッシュテーブルを用いる実際的によい方法を暗示している。