分散環境での L_1 距離ベース Locality-Sensitive Hashing のリモートアクセス回数削減

小栗　正之†　　古賀　久志†　　渡辺　俊典†

† 電気通信大学 大学院 情報システム学研究科
〒 182–8585 東京都調布市調布ヶ丘 1-5-1
E-mail: †{oguri,koga,watanabe}@sd.is.uec.ac.jp

あらまし　Locality-Sensitive Hashing (LSH) は高次元データに対する近似最近接点探索アルゴリズムである．LSH は高速な最近接点探索が可能な反面，ハッシュテーブルを複数個使用するため空間計算量が非常に大きい．そこで，LSH を複数計算機で構成される分散環境で実現して大規模化する技術が必要になる．LSH を分散環境で実現することを考えると，単純には各ノードにハッシュテーブルを均等に固定数ずつ配置する手法が考えられる．しかし，この方法では検索時に全てのノードに対するリモートアクセスが発生し，通信がボトルネックとなる分散環境では応答時間が長くなる．本研究ではハッシュバケツの配置を工夫して，検索時のリモートアクセス回数を減少させる方式を提案する．具体的には同じデータを含む異なるハッシュテーブル上のハッシュバケツ群をなるべく同じノード上に配置する．こうすることで，クエリ処理時に 1 回のリモートアクセスで複数のハッシュバケツにアクセスできるのでリモートアクセス回数を削減できる．

キーワード　近傍探索，Locality-Sensitive Hashing，分散環境，リモートアクセス

Reduction of Remote Accesses in Distributed Locality-Sensitive Hashing

Masayuki OGURI†, Hisashi KOGA†, and Toshinori WATANABE†

† Graduate School of Information Systems, University of Electro–Communications 1–5–1, Chofugaoka, Chofu-shi, Tokyo, 182–8585 Japan
E-mail: †{oguri,koga,watanabe}@sd.is.uec.ac.jp

1. はじめに

最近接点探索はパターン認識や情報検索の基盤となる技術である．多次元データに対する最近接点探索では，R-tree [1] に代表される木構造インデックスを用いる手法が知られている．しかし，この手法は厳密に最近点を求められる反面，次元数に対し計算量が指数的に増加する．

Locality-Sensitive Hashing [2] [3] (以下 LSH) は厳密にではなく，近似的に最近接点を求めることで高次元データに対しても計算量を抑える確率的アルゴリズムである．LSH では類似しているデータ同士は高い確率で同じハッシュ値となるようなハッシュ関数を複数個用意し，検索時にはクエリと同じハッシュ値を持つハッシュバケツ (以下バケツ) 内のデータを最近接点の候補とする．類似度計算をこの候補に対してのみ行うことで，最近接点探索のオーバーヘッドを削減する．ハッシュ関数を複数個用意する理由は確率的な候補点の見落としを防ぐためである．LSH はコサイン距離，Jaccard 係数など様々な類似度に適用できる．本稿では d 次元ユークリッド空間における L_1 距離に対する LSH [3] を取り扱う（d は次元数）．これは LSH に関する初期の論文で提案され，その後，p-安定分布を用いた $L_p(0 < p \leq 2)$ 距離に対する LSH [4] も提案された．LSH では高速な最近接点の探索が可能な反面，ハッシュテーブルが複数個必要なため，空間計算量が大きい．そして，探索精度を維持するためにはデータの次元数が増大するとハッシュテーブルの数も増やす必要がある．そこで，LSH を複数の計算機で構成される分散環境で実現して大規模化する技術が研究されている [5] [6] [7].

本研究では，LSH の複数個（l 個）のハッシュテーブルを n 個の計算機（以下ノード）に分散する状況を対象とする．この環境で単純に LSH を実現する手法としては，1 個のノードに $\frac{l}{n}$ 個ずつハッシュテーブルを格納する手法が考えられる．しかし，この方法ではクエリ処理時に l 個のハッシュテーブルにアクセスするために n 個の全ノードにアクセスしなければならない．ノードへのアクセスはネットワーク越しのリモートアクセ

スなので，通信がボトルネックとなる分散環境ではクエリへの応答時間が長くなってしまう．

本稿では，ノードへのリモートアクセス回数を減少させるLSHの実装方式を提案する．提案手法では1つのハッシュテーブルを1つのノードに配置せず，同じデータを含む異なるハッシュテーブル上のバケツ群をなるべく同じノード上に配置する．こうすることで，クエリ処理時には，1回のリモートアクセスで複数バケツにアクセスできるので，リモートアクセス回数を削減できる．人工データを用いたシミュレーションにより，提案手法が上述した単純な実現方法 (本稿では以下単純手法と呼ぶ) と比較してクエリ処理時のリモートアクセス回数を大幅に減少させる能力を有することを確認した．

本稿の構成は以下のようになる．2節で既存手法である L_1 距離ベースのLSH [3] を紹介する．3節で想定する分散環境を説明し，4節で提案手法を述べ，5節でシミュレーション結果を報告する．6節で関連研究を紹介し，最後に7節で結論と今後の研究の方向性について述べる．

2. L_1 距離ベースのLSH [3]

本節では，d 次元ユークリッド空間における L_1 距離ベースLSHについて説明する．次節以降でこのアルゴリズムを分散環境で実現する方式を論じる．

2.1 LSH関数

d 次元データ $p = (x_1, x_2, \cdots, x_d)$ とし，データ集合 $P = \{p_1, p_2, \cdots, p_n\}$ とする．ここではデータの座標値は正整数であるとする．そして，C を P のデータの中で座標値の最大値 $C = \max_{x_i \in p_j, p_j \in P}(x_i)$ とする．つまり，すべてのデータは辺長 C の d 次元超立方体内に存在する．

L_1 距離に対するLSHはこの超立方体を空間軸と直交するランダムに選択した k 個の空間分割面で分割することで実現され，分割により生成されたセル（格子）がハッシュバケツとなる．ここで空間分割面のランダムな選択は1から dC の範囲の整数をランダムに選択することで実現される．選択された整数を Z とすると，

$$\lceil \frac{Z}{C} \rceil \text{ 次元の値} = (Z-1) \bmod C + 1$$

となる超平面を選択したことになる．ハッシュ値はデータの座標と各超平面の座標との大小関係で決定され，データの座標が超平面の座標より小さければ0，超平面の座標以上であれば1を割り当てることで k ビットで表現される．図1は，$d = 2$, $C = 5$, 分割面数 $k = 2$ で空間分割面を $\{x = 3, y = 2\}$ としたときの空間の分割のされ方及び各セルのハッシュ値を表す．例えば点 (1,3) のハッシュ値は x 座標が3より小さく，y 座標が2より大きいので01となる．分割面数 k はパラメータである．

ハッシュの構成方法より L_1 距離が近いデータ同士ほどその間に空間分割面が配置される確率が低く，同じハッシュ値をとる確率が高い．しかし，空間分割のされ方によっては，空間的に近い点同士が同じセルに入らない (異なるハッシュ値をとる) 場合もあるため，LSHでは空間分割面が異なる複数個 (l 個) のハッシュ関数 $g_1(), \cdots, g_l()$ を用意する．例えば，図1の点Aと点Bは距離が近いが，Aを含むセルとBを含むセルは異なる．分割面が異なるハッシュ関数を複数あれば，AとBはどれかのハッシュ関数で同じバケツに入るようになる．つまり，複数のハッシュ関数により近接点の見落としを防止できる．

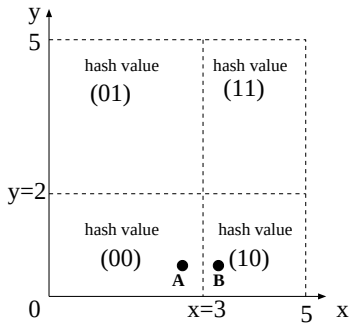

図1　空間の分割とハッシュ値の対応

2.2 最近接点探索

LSHによる最近接点探索はハッシュテーブルを生成する前処理の過程と近似最近接点探索を行う過程から成る．

- ハッシュテーブルの生成

Step 1. 空間分割面をランダムに選択した l 個のLSH関数 $g_1(), \cdots, g_l()$ を作る．

Step 2. 各LSH関数を P の全データに適用し，ハッシュテーブル $T_1, \cdots, T_l$ を作る．

- クエリに対する最近接点探索

Step 1. クエリ q に対し各LSH関数のハッシュ値 $g_1(q), \cdots, g_l(q)$ を計算する．

Step 2. q の各ハッシュ値に対応するバケツに格納されているデータを最近接点の候補とする．そして q と各候補間の距離を計算し，最も距離の近い点を最近接点として返す．

LSHでは最近接点をクエリと同じバケツに入った候補点にしぼって探索するので，高速な探索が実現できる．

3. 想定する分散環境

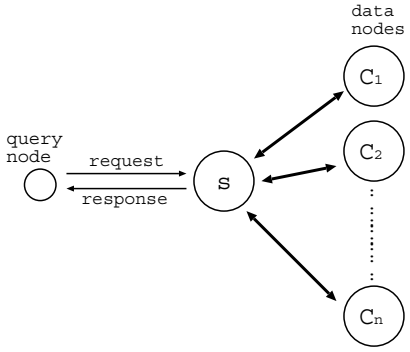

図2　想定分散環境

本稿が想定する分散環境を以下に述べる．本研究では，図2

に示すクライアント・サーバ型の探索システムを想定する．探索サーバは

（1） LSH における l 個のハッシュテーブルを分散して格納する n 個のデータノード $c_1, c_2, \cdots, c_n$: これらのノードは安定しており，故障による離脱や入れ替えはないものとする．

（2） クエリを受け付けるサーバノード s: s は c_1 から c_n のノードのアドレスを知っており相互に通信が可能である．

の 2 つの構成要素で構成される．

s はクライアントから最近接点探索リクエストを受け付けると，データノードと通信して最近接点を探索し結果をクライアントに返す．探索サーバが広域分散環境で実現され，ノード内の計算速度に対してノード間の通信速度がボトルネックになる環境を想定する．また，データノードの台数 n は P2P 環境のように数千台や数万台の規模ではなく，数台から数十台であり，とくにハッシュテーブル数より小さい (つまり $n \leqq l$) となることを仮定する．

本稿では s とデータノード間のアクセスのことをリモートアクセスと呼び，その回数削減を目指す．リモートアクセス回数を減らすだけであれば，全データを 1 つのデータノードに格納するのが最適解である．しかし，これでは特定のノードのみ空間計算量が増加するので，LSH を分散化する意義が失われる．従って，各データノードが保持するデータ数を均等に保ちながら，リモートアクセス回数を削減することが目標となる．1 節で述べた単純手法では，データノードが保持するデータ数は均等であるが，クエリ処理の度に多数 (n 回) のリモートアクセスが発生する．

4. 提 案 手 法

本節では，本研究の提案手法について述べる．

4.1 基本アイデア

以下では LSH における i 番目のハッシュテーブル T_i のハッシュ値 v のバケツを (i, v) と記述する．LSH ではクエリ q に対する最近接点探索では $(1, g_1(q)), (2, g_2(q)), \cdots, (l, g_l(q))$ の l 個のバケツがアクセスされる．従って，これらのバケツ群が同じノード上に存在すれば，1 回のノードアクセスで複数のバケツにアクセスできリモートアクセス回数を減らせる．提案手法では上記のアイデアを実現するために，同一のデータを含むバケツが同じハッシュ値をとるようなハッシュ関数 BH を構築し，BH のハッシュ値に基づいてバケツをノードへ割り当てる．以降では，

（1） 4.2 節でハッシュ関数 BH の構成方法

（2） 4.3 節で BH のハッシュ値に基くバケツのノードへの割り当て

について詳しく述べる．なお，以下ではデータに対するハッシュとバケツに対するハッシュを区別するため，BH をバケツハッシュ関数，BH のハッシュ値をバケツハッシュ値と呼ぶ．

4.2 バケツハッシュ関数 BH

同一のデータ x を含むバケツ群は d 次元のユークリッド空間上では互いに近接しており，それぞれの重心も互いに近い．例えば，図 3(a),(b) はそれぞれ，2 つのハッシュ関数 g_1,g_2 による空間分割を示す．x を内包するセルが x を格納する 2 つのバケツ $(1, g_1(x))$,$(2, g_2(x))$ を表す．$(1, g_1(x))$ と $(2, g_2(x))$ は共に x を含むので空間的に近接し，それらの重心も近い．この性質から，重心点が近いバケツが同じハッシュ値となるハッシュ関数を作れれば，目的の BH を構成できる．

これは，空間的に近い (重心) 点が同じハッシュ値を取るハッシュ関数であり，まさに LSH 関数に他ならない．そこで，空間分割面数を k より減らした LSH 関数を BH とする．BH の空間分割面数を $k_b (< k)$ とおく．分割面数を減少させることで，BH のセルがデータに対するハッシュ関数 g_i のセルより大きくなり，同じデータを含むバケツ群の重心座標が同一のバケツハッシュ値を取ることが期待できる．図 3 の例では図 (c) が BH による空間分割を表す．BH では空間が粗く分割されたことで，$(1, g_1(x))$ と $(2, g_2(x))$ の重心座標が同一のバケツハッシュ値になる．

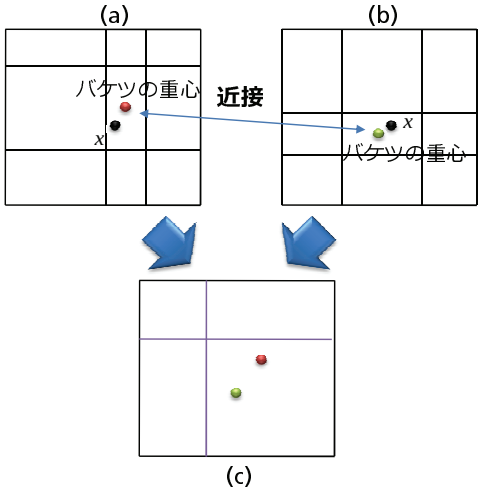

図 3 異なるハッシュテーブル上の同じ点を含むバケツ

4.3 バケツのノードへの割り当て

BH の分割面数は k_b なので，バケツハッシュ値の値域は $0 \leqq BH(x) \leqq 2^{k_b} - 1$ となる．すなわち，BH はバケツを $0 \leqq BH(x) \leqq 2^{k_b} - 1$ の区間に射影する．従って，$0 \leqq BH(x) \leqq 2^{k_b} - 1$ の区間を n 個の区間に分割すれば，バケツ (バケツハッシュ値) を n 個のノードに対応させられる．

3 節で述べたように，各ノードに割り当てられるデータ数は均等であることが望ましい．しかし，一般的にはデータ集合 P には分布の偏りがあり，P のデータを含むバケツのバケツハッシュ値も一様に分布しない．従って，$0 \leqq BH(x) \leqq 2^{k_b} - 1$ の区間を n 個に等分割しても，各ノードに割り当てられるデータ数は均等にならない．P の分布の偏りはデータごとに異なるので，本研究ではランダムサンプリングにより P の分布を学習し，その結果から $0 \leqq BH(x) \leqq 2^{k_b} - 1$ の区間の分割方法を決定する．以下に区間分割アルゴリズムを示す．

（1） P から α%のデータをランダムサンプリングにより取り出し，部分データ集合 S とする．5 節の実験では $\alpha = 10$ % とした．

（2） S の各要素 s を BH に適用しハッシュ値 $BH(s)$ を計算する．ハッシュ値 $BH(s)$ の集合を V とする．

$$V = \{BH(s) | s \in S\}.$$

（3） V の要素をソートする．そして，各ノードに $\frac{|S|}{n}$ 個ずつ要素を割り当て，各ノードに割り当てられる区間を決定する．具体的には，i 番目のデータノード c_i $(1 \leqq i \leqq n)$ には V の $\frac{(i-1)*|S|}{n}$ 番目の値より真に大かつ $\frac{i*|S|}{n}$ 番目の値以下の区間を割り当てる．

4.4 探索サーバにおける処理手順

探索サーバにおけるデータの登録とクエリ処理の手順はそれぞれ以下のようになる．

4.4.1 データの登録

データ p のデータノードへの登録は以下の手順で実現される．

Step 1. s が p の l 個のハッシュ値 $g_1(p), g_2(p), \cdots, g_l(p)$ を計算する．

Step 2. s が l 個のバケツ $(i, g_i(p))$ $(1 \leqq i \leqq l)$ の重心座標を計算する．バケツの重心座標をバケツハッシュ関数 BH に適用し，バケツハッシュ値 $BL((i, g_i(p))$ を得る．バケツハッシュ値 $BL((i, g_i(p))$ と 4.3 節で述べたバケツハッシュ値のノードへの割り当てから p を格納するデータノードがこの時点で決定する．

Step 3. s がデータノードへ p を格納することを要求する．要求を受けたデータノードが p を格納する．

Step 3 では，s はデータノードに { ハッシュ関数の番号 $i, g_i(p), p$} の 3 つ組を送り，データノードはバケツ $(i, g_i(p))$ へ p を格納する．なお，データノード内部では，バケツは (ハッシュ番号) と (ハッシュ値) をキーとするハッシュテーブルに登録され，任意のバケツに高速にアクセスできる．

4.4.2 クエリ処理

クエリ q に対する最近接点探索の処理は以下のようになる．

Step 1. s が q の l 個のハッシュ値 $g_1(q), g_2(q), \cdots, g_l(q)$ を計算する．

Step 2. s が l 個のバケツ $(i, g_i(q))$ $(1 \leqq i \leqq l)$ の重心座標を計算する．バケツの重心座標をバケツハッシュ関数 BH に適用し，バケツハッシュ値 $BL((i, g_i(q))$ を得る．得られたバケツハッシュ値と 4.3 節で述べたバケツハッシュ値のノードへの割り当てから，s がアクセスすべきデータノードが決定される．

Step 3. s がアクセスすべきデータノードへ { ハッシュ関数番号 $i, g_i(q)$} の 2 つ組を送る．アクセス対象のバケツが同じノード上に複数存在する場合は，2 つ組を一度に複数送り，複数バケツへのアクセスを 1 回のリモートアクセスで完了する．データノードはバケツ $(i, g_i(q))$ に含まれるデータを q の最近接点の候補として s へ返す．

Step 4. s が，データノードから返された各候補点と q との距離を計算し，q から最も距離が短い点を最近接点としてクライアントに解答する．

4.5 提案手法の探索精度

提案手法がアクセスするバケツは，1 台の計算機で実装した通常の LSH と全く同じである．従って，提案手法は 1 台の計算機で実装した通常の LSH と同一の探索結果を返す．この性質は単純手法に関しても成立するため，5 節のシミュレーションでは探索精度は評価対象としない．

5. 実験による提案手法の評価

人工データを用いたシミュレーションにより，提案手法の実験を評価する．評価項目は

- クエリ処理時のリモートアクセス回数
- データノード毎の登録データ数

である．

5.1 実験環境

20 次元ユークリッド空間において，各次元の座標値が 1 以上 1000 未満を満足する辺長 1000 の超立方体内に 8 個の正規分布から 1250 個ずつ点を生成し，合計 10000 点からなるデータ集合 P を生成した．正規分布の重心は各次元の座標値 [100,900] の範囲でランダムに選択した．また，各次元の標準偏差 σ は等しく 60 である．4.3 節で説明したバケツのノードへの割り当てを決めるためのランダムサンプリングでは P から 10%のデータを取り出した．P をデータノードに登録した後，上述の 8 個の正規分布から 50 点ずつ合計 400 点のクエリを生成し最近接点探索を行った．

LSH については，分割面数 $k = 192$，ハッシュ関数の数 $l = 20$，バケツハッシュ関数 BH の分割面数 $k_b = 160$ と固定する．

5.2 実験結果

リモートアクセス回数:

ノード数 n を $n = 5, 10, 15, 20$ と変化させた時の，400 点のクエリに対する合計のリモートアクセス回数を図 4 に示す．横軸はノード数 n で縦軸が総リモートアクセス回数である．なお，測定はデータセットを 10 個生成し，それぞれに提案手法を 10 回適用して合計 100 回行い，その平均値をプロットした．比較のために，1 章で述べた単純手法によるリモートアクセス回数を併せて示す．グラフでは実線が提案手法，破線が単純手法である．$l \geqq n$ なので単純手法ではすべてのノードが少なくとも 1 つの LSH のハッシュテーブルを持つ．従って，1 つのクエリ処理で全ノードへのアクセスが発生し，400 点のクエリに対する総リモートアクセス回数は $400n$ となる．

図 4 よりすべての n に対して，提案手法は単純手法よりリモートアクセス回数を大幅に削減できている．提案手法のリモートアクセス回数は $n = 5$ の時，単純手法の 50%程度，$n = 20$ の時，単純手法の 26%程度である．つまり，n が大きくなるほど単純手法に対するアドバンテージが大きい．また，単純手法ではリモートアクセス回数が n に関してリニアに増加するが，提案手法ではサブリニアな増加である．一般にノード数 n は，登録されるデータ数が増加し空間計算量が大きくなると増やす必要があるので，提案手法は大規模な LSH を実現するのに適する．

ノードごとの登録データ数:

図 5 に $n = 10$ の時のノードごとの登録データ数をデータ数の多い順に示す．データ集合 P のサイズ $|P| = 10000$ であり，$l = 20$ なので，データはのべ 20 万回登録される．従って，1

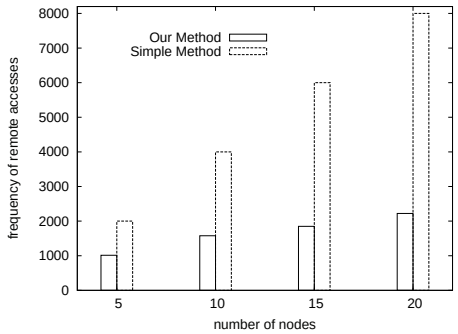

図 4　リモートアクセス発生数

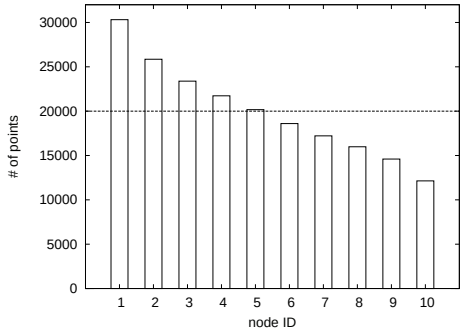

図 5　ノードごとのデータ数

つのノードに $\frac{|P|*l}{n} = 20000$ 個ずつ登録されている状態が理想である (図の破線)．しかし，実際にはデータ数最大のノードのデータ数が約 30000，データ数最小のノードのデータ数が約 12000 となり，最大のノードと最小のノードで 2.5 倍の差が生じた．我々はこの結果を，登録データ数の均等化がある程度出来ているがまだ改善されるべき状態である，と考えている．

登録データ数の偏りが発生した理由は次のように推測される．本稿では，各ノードが担当するバケツハッシュ値の範囲を，P からサンプリングしたデータ点の座標に対する BH のハッシュ値から学習して決定した．しかし実際には，データが登録されるノードは，データ点そのものではなく，データ点が入るバケツ重心点の座標のバケツハッシュ値によって決定される．BH では分割面数が少ないので，確率的に大きなサイズのセルが生成され，バケツ重心とバケツ内のデータ点の座標とが大きく異なることがある．この現象により，データ点座標を用いた学習の効果が限定的になったと考えられる．バケツ重心点座標を用いて学習することにより，登録データ数の偏りを軽減することが今後の課題である．

6. 関 連 研 究

LSH の分散環境での実現に関する既存研究は [5] [6] [7] がある．いずれも LSH を P2P ネットワークにおける類似検索に適用している．

本研究と最も近い既存研究は Haghani らによる [7] である．この研究では，1 個のハッシュテーブル内の類似データを含むバケツ群を同一ノードに配置し，クエリ処理時にはクエリが属するバケツに近接したバケツも調べる手法を提案した．この手法では，1 回のリモートアクセスで 1 つのハッシュテーブルから多くの最近接点候補が得られる．その結果，必要なハッシュテーブル数が減らせるので，リモートアクセス回数も削減できる．本研究では，異なるハッシュテーブルに属するバケツでも類似データを含むならば同一ノードに配置するよう試みる点が [7] と異なる．

Hua ら [6] は，P2P ファイルシステムにおけるデータ分布の偏りに着目した．データ密度が高い空間に対応するバケツは多くの点を含み，クエリ処理時に大量の候補点を生成して応答時間を増加させる．この問題を解決するために 1 つのバケツに格納できるデータ数の上限を設け，上限を超えたデータを隣接バケツに配置する手法を提案した．

7. ま と め

本研究では，高次元の近似最近接点探索アルゴリズム LSH を複数台 (n 台) の計算機に分散して実現する状況を考えた．LSH は複数個 (l 個) のハッシュテーブルを用いるため空間使用量が大きく，大規模化するには複数ハッシュテーブルを複数計算機上に配置するアプローチが有望である．

分散環境で LSH を単純に実装する手法 (単純手法) としては，各ノードに LSH のハッシュテーブルを等しく $\frac{l}{n}$ 個だけ配置する手法が考えられる．しかし，この単純方法ではクエリの度に全ノードへのアクセスが発生し，通信がボトルネックとなる分散環境では応答時間が長くなる．

そこで本稿では，複数のハッシュテーブル上の同一データを含むバケツ群をできるだけ同じノードに配置することで，リモートアクセス回数を抑える手法を提案した．具体的には，同じデータを含むバケツ群が空間的に近接することに着目し，バケツ重心座標を LSH に適用して空間的に近接したバケツが同じバケツハッシュ値を取るようにした．そして，バケツハッシュ値からバケツを割り当てるノードを決定し，同一データを含むバケツ群が同一ノードに割り当てられるようにした．人工データを用いたシミュレーションにより，提案手法が単純手法より，クエリ処理時のリモートアクセス回数を 50%以上削減できることを示した．

一方で，提案手法では単純手法に比べクエリ処理時にバケツハッシュ値を計算する計算オーバーヘッドが発生する．今回のシミュレーションでは計算オーバーヘッドを評価できないので，今後は提案手法を実機に実装して性能評価を行う．また，ノード毎の登録データ数をより均等にする手法の設計も進める．

謝辞　本研究は科学研究費基盤研究（C）課題番号 21500008 の支援を受けて行った．

文　　献

[1] A. Guttman, “R-trees: A dynamic index structure for spatial searching”, In Proc. ACM Conference on Management of Data (SIGMOD), pp.47-57, 1984.

[2] P. Indyk, R. Motwani, “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality”, In Proc. 30th ACM Symposium on Theory of Computing, pp.604-613, 1998.

[3] A. Gionis, P. Indyk, R. Motwani, “Similarity Search in High Dimensions via Hashing”, In Proc. 25th VLDB Conference,

pp.518-528, 1999.

[4] N. Immorlica, P. Indyk, V. S. Mirrokni, "Locality-Sensitive Hashing Scheme based on p-Stable Distributions", in Proc. 21th ACM symposium on Computational Geometry, pp.253-262, 2004.

[5] M. Bawa, T. Condie, P. Ganesan, "LSH Forest: Self-Tuning Indexes for Similarity Search", In Proc. 14th International World Wide Web Conference, pp.651-660, 2005.

[6] Y. Hua, B. Xiao, D. Feng, B. Yu: "Bounded LSH for Similarity Search in Peer-to-Peer File Systems", In Proc. 2008 International Conference on Parallel Processing, pp.644-651, 2008.

[7] P. Haghani, S. Michel, P. Cudr-Mauroux, K. Aberer, "LSH At Large - Distributed KNN Search in High Dimensions", In Proc. 11th International Workshop on the Web and Databases (WebDB), 2008.