# 連想配列型分散データ構造におけるキーの局所性を考慮した負荷分散

岡 敏生[†] 森川博之[†] 青山友紀[††]

特徴量に基づく類似検索や3次元位置情報管理システム等では，クエリ解決において局所的なデータ集合に対して処理を行う必要性が生じる．このような場合，データの配置がシステムの性能を決定する重要な要素となる．本稿では大規模分散環境において前述の処理を行うのに適した負荷分散について述べる．ハッシュテーブルに基づく従来の負荷分散ではキーの局所性が損なわれるため，システムの効率性が失われてしまう．そこでブロックリダイレクションという手法を導入し，キーの局所性を維持した負荷の分散を実現した．以下に示した手法ではシステムの平均負荷がほぼ一定であり，かつその平均負荷が定数倍の範囲で事前に分かっている場合，任意のデータ分布に対して，最も負荷の重いホストにかかる負荷が高い確率で平均負荷の $\Theta(\log\log N/\log l)$ 倍になるという特性が得られる．他方，システムの平均負荷が事前に分かっていない場合についても，前述の特性の保証は得られないものの，優れた負荷分散が図られることをシミュレーションによって検証した．

# Locality-preserving Load Balancing for Distributed Associative Data Structures

Toshio Oka,[†] Hiroyuki Morikawa[†] and Tomonori Aoyama[††]

Query resolution in similarity search and geographical information processing entails data access to a specific region in a dataset. In these types of systems, data placement is one of the dominant factors for the throughput of data processing. This paper presents a load balance scheme that is suitable for locality-sensitive data access in distributed environments. The scheme is based on block-wise redirection and it achieves a good asymptotic property, while preserving the locality of data placement. The load of the most-loaded host is $\Theta(\log\log N/\log l)$ for arbitrary data distribution when the average load of the system is nearly constant and the load is known in advance. The effectiveness of the scheme holds even without the prior knowledge on the average load, although the asymptotic property is not guaranteed in this case.

## 1. はじめに

本稿ではキーの局所性を意識した分散連想配列の負荷分散アルゴリズムについて述べる．ここでいう分散連想配列とは大規模分散環境で連想配列の機能を提供するものを指すこととする．以下ではデータの格納位置の決定方法を除いて，分散ハッシュテーブル（Distributed Hash Table）[8),11)~13),15)] の仕組みをほぼ踏襲してこの分散連想配列を実現する．

Chord[13)] や Content Addressable Network（CAN）[11)] といった分散ハッシュテーブルでは与えられたキー（key）とそれに対応する値（value）をあらかじめ格納しておき，クライアントがキーを指定したとき，それに対応する値を返すという単純な機能を提供する．分散ハッシュテーブルは一般的にいわゆる peer-to-peer 型の構造を持っており，システム内の特定の部位に処理のボトルネックや単一障害点が生じないように設計されている．またホスト，ネットワークの障害やホスト参加，離脱等によって動的に参加ホストの構成に変化が生じた場合でも適切に動作するといった性質もあわせ持っている．

単純なデータルックアップを提供するこれらの仕組みに対して，近年，分散ハッシュテーブルの構造を利用しながら，付加的な機能を提供しようという試みがいくつか提案されている．たとえば pLSI[14)] では多次元分散ハッシュテーブルである CAN の基本的な構造を踏襲しつつ類似検索の機能を提供している．pLSI では特徴量に基づいて当該文書を CAN の $d$ 次元空

† 東京大学大学院新領域創成科学研究科
Graduate School of Frontier Sciences, the University of Tokyo
†† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, the University of Tokyo

間にマッピングしている．このとき類似文書が同一のホストもしくは論理トポロジ上の近隣ホストに格納される可能性が高くなるため，ハッシュ関数に基づいてランダムにデータ配置する場合に比べて検索の効率が格段に向上する．そのほかにも属性検索に分散ハッシュテーブルの枠組みを利用する動きもある[2]．属性検索では複数属性に関して属性名と属性値（の範囲）に該当するオブジェクトを見つけ出すといったことを行う．このとき，同じ属性について近い属性値を持つオブジェクトがなるべく同一のホストに格納されるようにすれば，先ほどと同様の理由によって検索効率の向上が図られる．

このようなシステムにおける課題は，ハッシュ関数による負荷分散を用いないため一般的にデータ分布の偏りが負荷の偏りにつながってしまうことである☆．したがって円滑に処理が行われるよう，タスクを均等に分散させる必要がある．

本稿で扱うのは単純なハッシュテーブルの負荷分散ではなく，主としてこのようなシステムにおける負荷分散である．以下ではキーの局所性という観点に注目したうえで効果的な負荷分散手法を示す．たとえば位置座標に基づいてオブジェクトを管理する 3 次元実空間情報管理システムにおいてハッシュテーブルのようにデータをランダム化して格納してしまうと，あるエリアに属するオブジェクトのみを収集したい場合でもデータ全体を探索する必要が生じてしまう．したがって，同時にアクセスされるキーには局所性があることを考慮に入れて，キーの類似したオブジェクトがなるべく同一のホストに配置されるように負荷分散することが望ましい．

以下ではブロックリダイレクションという手法を導入し，キーの局所性を意識した負荷分散を行う．キーの類似したデータどうしをまとめてブロックを構成し，他のホストへリダイレクトするというのが基本的な考え方である．提示した手法は局所性を意識しているという利点だけでなく，優れた負荷分散特性を有している．本手法では，ある定数 $\lambda \geq 1$ について，システムの平均負荷 $L_m$ がつねに $L_i \leq L_m \leq \lambda L_i$ を満たし，さらにこの $L_i$ が既知であるとき，任意のデータ分布に関して最もデータ格納数の多いホストの負荷が $\Theta(\log\log N/\log l)$ となることが確率的に保証される（$N$ はシステム内のホストの総数，$l$ は後述するシステムパラメータ）．したがって，どのようなデータの分布であろうとデータ格納に関するホストの負荷は基本的に大きく変化しない．後述するリダイレクションポインタの数はデータ分布に依存することとなるが，システム全体でのリダイレクションポインタの総数は $\Theta(N)$ である．ここで各データへのアクセスパターンが定常的であると仮定すると，データ格納数をアクセス頻度に置き換えることによって任意のデータ分布，アクセスパターンについても同様の負荷分散特性が得られる．

本稿の構成は以下のとおり．まず 2 章では対象とする環境および目的について述べ，求められる負荷分散手法の要件について触れる．次いで 3 章でブロックリダイレクションという負荷分散手法の仕組みを示す．そして 4 章では前述の $L_i$ が未知である場合について負荷分散がどのようになるかシミュレーションによって検証する．5 章では関連する研究を紹介し，最後に 6 章でまとめを行う．

## 2. キーの局所性を維持した負荷分散

### 2.1 想定する環境

本稿では数十台から数百万台の一般的な PC もしくはワークステーションによって構成される大規模分散システムを想定する．構成するホストには当該ホストのハードディスクの故障，OS 等のダウンやネットワークの部分的切断等の可能性がある．また中長期的にはシステム規模が変化する可能性も想定する．これらの理由により，システムに参加しているホストの集合は動的に変化しうるものとする．

そしてそれらのホスト群は，システム内の特定の部位に処理のボトルネックや単一障害点が生じてしまわないように完全自律分散的に動作しているものとする．これらのホストがすべて LAN 内に存在するか，広域ネットワークによって接続されているかについては特に限定しない．

### 2.2 分散ハッシュテーブル

2.1 節で示した状況下で適切にデータのルックアップの機能を提供するためのものとして提案されているのが分散ハッシュテーブルとよばれるものである．代表的なものとして Chord，Pastry [12]，Tapestry [15]，Koorde [8]，CAN 等があるが，ここでは $d$ 次元キーを元にしたデータルックアップを提供する Content-Addressable Network（CAN）について概説する．

CAN では各データをどのホストに格納するかという割当てを $d$ 次元座標を通じて行っている．$[0,1] \times [0,1] \times ... \times [0,1]$ からなる $d$ 次元空間 $U$ は**図 1** に示したように分割され，いずれかのホストに割り当てら

☆ このようなシステムのなかには，ハッシュ関数を用いながらも特定のキーに負荷が集中している場合もある．

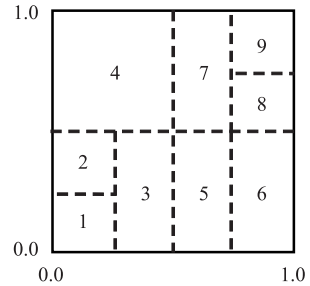


**図 1** Content-addressable network（次元 $d = 2$）
Fig. 1 Content-addressable network ($d = 2$).

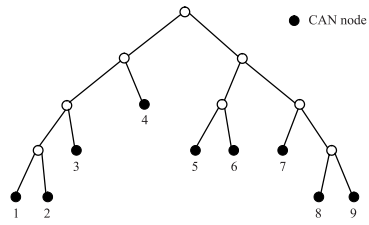


**図 2** 分割木
Fig. 2 Partition tree.

れる（以下，ホストのゾーンと呼ぶ）．図では空間 $U$ は 9 つのゾーンに分割されている．

他方，CAN に格納される（キー，値）のペア（以下，オブジェクトと呼ぶ）は空間 $U$ 内の点にマッピングされる．マッピングされた点 $P(\boldsymbol{k})$ がホスト $A$ のゾーン $Z_A$ に含まれている場合，この（キー，値）のペアをホスト $A$ に格納するものとする．本来 CAN ではキーにハッシュ関数をかけることによってランダムな座標にマッピングしているが，本稿ではハッシュ関数を用いないで空間 $U$ にマッピングする．

ホストに対してどのようにゾーンを割り当てるかは**図 2** に示した分割木（partition tree）という概念によって理解することができる．システムにホストが参加するたびにこの二分木の葉節点は分割され，子節点として新しく 2 つの葉節点が生成される．片側の節点は元々のホストに再割当てされ，もう片側の節点は新規参加ホストに割り当てられる．ホストがシステムから離脱する場合には逆の操作，つまり 2 つの葉節点を除去し，もう片方のホストにはその親節点を割り当てる．図 1 と図 2 の対応関係に注意されたい．ホストにどの節点を割り当てるかはホストのランダムな ID に基づいて決まるため，分割木はほぼ平衡している．実際の CAN ではより平衡したものとするために，小さな修正が加えられている．

CAN では空間上の隣接ホスト間の通信を行うため，互いに接しているゾーンは相手ホストの（IP アドレス，ポート番号）を管理している．これによって$\boldsymbol{k}$を知る任意の参加ホストは $O(d \times N^{1/d})$ ホップでホスト $A$ にメッセージを送ることができる（$N$ は CAN の参加ホスト数）．

### 2.3 対象とするアプリケーション

本稿で主として対象とするアプリケーションは，各オブジェクト間に近接性（proximity）という概念が存在し，近接性の高いオブジェクトへのアクセスが同時に発生する確率が高いという性質を持ったものである．そして互いにキーが近接するオブジェクトどうしで集合を形成できる場合，効果的に動作する．たとえばキーの間に三角不等式 $d(x, y) + d(y, z) \geq d(x, z)$ が成り立つ場合，$x$ と $y$ が近く $y$ と $z$ が近い場合，$x$ と $z$ も近くなる傾向があり，このような集合が得られやすい（ただし距離 $d(\cdot)$ はつねに 0 以上とする）．

本節では当該手法が特に有効であるアプリケーションをいくつか例示する．代表的なものとしてまず 3 次元空間内のオブジェクトの情報管理があげられる．3 次元空間内の位置情報に基づいてオブジェクトを管理するような場合，位置がある範囲内にあるオブジェクトを対象に処理を行うといったことが考えられる．そのような場合，先ほどの 3 次元 CAN において該当するホストにだけクエリを送れば目的を達成できる．ただし，空間の特定のエリアにオブジェクトが大量に存在する場合，一部のホストに負荷が集中してしまうため，アクセスの局所性を意識して負荷分散を行う必要がある．

その他の例として情報収集分野があげられる．pLSI では LSI [7] とよばれる次元削減手法を用いる等して文書の特徴量ベクトルを CAN 上のホストとマッピングしている．pLSI では同一のホスト，あるいは論理トポロジが近隣のホストに類似文書が格納されるようになるため，検索が容易に行えるようになる．

近似最近傍点探索も適用アプリケーションの 1 つである．Indyk らは Locality Sensitive Hashing（LSH）とよばれる仕組みによって $L_p$ ノルム（$p \in (0, 2]$）を与えられた高次元の計量ベクトル空間等においてクエリ点の最近傍にある点を近似的に求めるアルゴリズムを実現している[6]．LSH では先の例と異なり 1 次元のハッシュテーブルにデータを格納するため，分散化する場合には Chord 等の分散ハッシュテーブルに適用することとなる．

類似オブジェクトの検索以外にも属性検索も本手法の適用対象となる．複数の属性の属性値に基づいてオブジェクトを指定する場合，効率的に検索するには属性値が近いオブジェクトはなるべく同一のホストに格

納されている必要がある．したがって他のアプリケーションと同様，キーの局所性を意識した負荷分散が必要とされる．

本稿で示す手法自体はキーの局所性を意識する必要のない環境下でも十分有効であり，実際には適用可能範囲はより広いものとなる．

### 2.4 負荷分散に求められる要件

本節では対象アプリケーションにおいて負荷分散に求められる要件について述べる．求められる要件は以下のとおりである．

- キーの局所性を維持した負荷分散
- 完全分散性
- スケーラビリティ
- 最悪負荷の保証

まず，すでに触れたように，キーが類似したデータを同時にアクセスするといった状況下では負荷分散したのちもキーの局所性が維持される必要がある．

次にあげた完全分散性は負荷分散アルゴリズムの実現方法に求められる要件である．負荷分散の 1 つの実現方法として，クエリを一カ所にまとめ集中的にタスクの分配を行うといった方法が考えられる．このような方法は実際しばしば採用される手法であるが，分散ハッシュテーブルの目的から考えると好ましい解決方法ではない．このような手法はあるシステム規模を超えた段階でパフォーマンスのボトルネックとなってしまうため，タスクの分配は分散的になされることが求められる．

その次にあげたスケーラビリティも同様の理由である．分散ハッシュテーブルが大規模なシステムを対象としたものであるため，負荷分散アルゴリズム自体もシステムの規模に適応できるものでないとまったく意味がない．そのため負荷分散に必要な計算量および通信量はシステムの規模が大きくなっても許容可能なものでなくてはならない．

最後にあげた負荷分散特性の保証はシステムの可用性等の観点から重要となる．最も負荷のかかるホストの負荷を抑制できなければ，サービスの提供にかかる時間が許容できないほど長くなってしまったり，分散システムの維持自体ができなくなったりしてしまう．負荷が最も重いホストの負荷を軽減することは非常に重要である．

上記にあげた 4 つの項目以外に考えられる要件として，各ホストの処理容量の非均一性への対処といったこともあげられるが，本稿では特にこれを取り扱わない．ホストの容量が均一でない場合の負荷分散については，仮想サーバ[10] というメカニズムによって解決するものとする．すなわち一般的なホストよりも処理容量の大きなホストは仮想的なサーバを複数立ち上げる．こうすることによって問題は単純に仮想サーバ間で負荷を均一化するという問題に帰着できることが分かる．

## 3. ブロックリダイレクション

### 3.1 基本概念

3 章ではブロックリダイレクションという手法の仕組みについて述べる．それに先だってここではブロックリダイレクションの基本的な概念を述べることとする．

ブロックリダイレクションとは，キーが近接したオブジェクトどうしでまとめて仮想的に大きなブロックを構成し，(キー，値）ペアとホストのマッピングをブロック単位で別のホストにやり直すものである．以下では，元々マッピングされていたホストを**リダイレクション元**，新たにマッピングされたホストを**リダイレクション先**と呼ぶことにする．リダイレクション元はリダイレクション先の ID と（IP，ポート番号），リダイレクトされたブロックのサイズを記憶する．ここで ID と（IP，ポート番号）をリダイレクションポインタと呼ぶこととする．ID さえあれば IP 等の情報はなくても動作するが，これによってクエリの解決を高速化することが可能になる．リダイレクション先がホスト全体から均一に選ばれるようにすることによって，負荷の均一化が行われる．

以下に，ブロックリダイレクションを用いた場合のデータルックアップの大まかな流れを示す．まずクライアントは，キーを担当するリダイレクション元のホストにリダイレクション先のホスト情報を問い合わせる．このときメッセージの伝達には Chord 等のメッセージルーティングを用いる．問合せを受けたリダイレクション元はリダイレクト先のホストの ID および（IP，ポート番号）を返す．クライアントはこの（IP，ポート番号）情報を用いてリダイレクト先と通信する．リダイレクト先がノードの離脱等によってすでに変更されている場合，リダイレクション先の ID を用いて新しいリダイレクション先と通信する．このときリダイレクション先として複数のホストが与えられることもある．多くのアプリケーションでは複数のホストが与えられたとき，すべてのホストと通信することになる．本稿で扱うブロックリダイレクションはあくまで仮想的なものであり，実際のデータのやりとりはリダイレクション元を経由せず，クライアントとリダイレクション先の間で直接行われる．

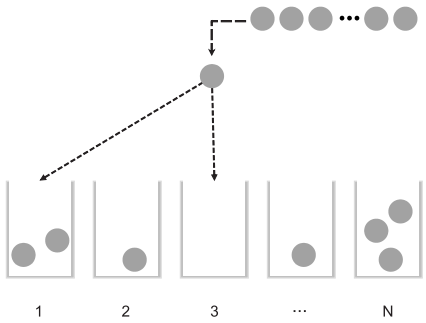


**図 3** bins and balls による過程
Fig. 3 Bins and balls.

### 3.2 リダイレクション先ホストの決定

本節ではリダイレクションにおいてどのようにリダイレクション先を選ぶかについて述べる．どのように選ぶかが負荷分散特性を決定する重要な要素である．筆者らは “bins and balls” というよく知られた仕組みを利用してリダイレクション先を選択する．

**bins and balls**

**図 3** に示したとおり，$N$ 個のビンがあるとする．以下の規則に従って $M(M \geq N)$ 個のボールを 1 つずつその中に入れてゆく．

(1) $N$ 個のビンからランダムに $l(l \geq 2)$ 個のビンを選択する．
(2) $l$ 個のビンのうち最もボールの少ないビンにボールを入れる．

この規則に則って 1 つずつボールを入れてゆくと，最もボールの多いビンには高い確率で $\log\log N/\log l + \Theta(M/N)$ 個のボールが入ることが知られている（ただし，$N \to \infty$）[1]．また，この際ビンの選択は一様である必要はなく，たとえば一部のビンが選ばれる確率が $\Theta(\log N/N)$ の場合でも $\Theta(1)$ のペナルティで同様の性質が得られることが知られている（各ビンの選ばれる確率の分布には制約があるので，詳細は参考文献 4) 参照のこと）．

ここでビンをホストに置き換え，ボールをブロックに置き換えて考えると，ブロックの均一なリダイレクション法が得られる．きわめて単純な操作によってきわめて良好な負荷分散特性が得られることが分かる．

### 3.3 目標ブロックサイズの決定

3.2 節の仕組みを適用するうえでまず解決しなくてはならないのは，目標とするブロックサイズ $S_t$ の決定である．ブロックサイズが小さすぎるとリダイレクションポインタの数が多くなってしまう．また 1 回のクエリ解決に関わるホストの数も大きくなってしまう．逆にブロックサイズが大きすぎると負荷分散特性が劣化してしまう．

そのような観点から目標ブロックサイズはシステムの平均負荷 $L_m = L/N$ を超えない，なるべく大きな値となるように設定する（$L$ は系内の総データ量とする）．

ここで，ある定数 $\lambda \geq 1$ について $L_i \leq L_m \leq \lambda L_m$ を満たす $L_i$ が既知であるとき，ブロックサイズは $L_i/2$ に固定する．実用上はこの $\lambda$ を小さな値に設定しておく必要がある．

他方，平均負荷について事前知識を持たない場合，システムの平均負荷を推定することとなる．問題は大規模分散環境では系内の総データ量を求めることは事実上不可能であるうえ，この目標ブロックサイズは動的に変化するものである点である．筆者らはサンプリングによって $L/N$ の値を推定することでこの問題を回避する．各ホストは 3.5 節で述べる手法を用いて定期的にランダムサンプリングを行い，$m$ 個のホストの負荷の標本平均 $l_a$ によって母平均 $L/N$ に代用する．各ホストの負荷がほぼ均等になっている状況では，少数のサンプリングでも十分な精度が得られる．ただし，このとき最悪負荷についての保証は成立しない．

### 3.4 ブロックサイズの調整

ブロックの構築は検索コストを決定する重要な要素であるが，適したブロックの構築方法は対象とするシステムに依存するので，本稿ではキーに依存しない最も単純なブロック構築方法を採用する．つまりあるリダイレクション元にマッピングされたデータは任意のブロックに入れてよいものとする．

あるリダイレクション元から $b$ $(b \geq 1)$ 個のブロックがリダイレクトされるとき，データはクライアントとリダイレクション先の間で直接やりとりされるため，必ずしも各ブロックのサイズは目標ブロックサイズに一致しない．したがって，以下の規則に従って定期的にブロックサイズを調整する．

(1) すべてのブロックサイズが $S_t$ 以下のとき
ブロックサイズの総和が $(b-1) \times S_t$ 以下となった段階でブロックを 1 つ減らし，別のブロックにデータを移動する．
(2) すべてのブロックサイズが $S_t$ 以上のとき
ブロックサイズの総和が $(b+1) \times S_t$ となった段階でブロックを 1 つ増やし，別のブロックからデータを移動する．
(3) $S_t$ より大きいもの小さいものが混在するとき
大きいブロックから小さいブロックにデータを移動する．

対象とするアプリケーションによっては，キーを意

識しないこのようなブロックの構成は必ずしも効率的ではない．そのような場合，ブロックの構成方法を変更することで検索コストを削減することができる．

### 3.5 ランダムホストの選択

3.2 節と 3.3 節で用いるランダムホストの選択について考察する．ただしすでに述べたとおり，最悪負荷の保証という面では “bins and balls” プロセスにおけるホスト選択は必ずしも完全に均等である必要はない．

ランダムホストの選択は LAN 内であれば比較的単純である．すべてのホストの IP 情報を LAN 内に分散配置された数台のサーバに登録し，そのサーバでランダムにホストを選択すればよいだけであるからである．しかし大規模広域システムを想定すると，そのような単純な方法の問題が浮かび上がってくる．まず少数のサーバに全体の安定性を依存することとなるため，システムの脆弱性が高まってしまうといった問題がある．それに加えて各ホストはすべての管理サーバに定期的に正常動作を伝えるメッセージ（heartbeat message）送信や 3.3 節で行う負荷のサンプリングの必要があるため，通信コストが大きなものとなってしまう．

そこで以下では分散的にランダムなホストを選択する仕組みを導入する．ここでは Chord を例にとって考えるが，Pastry や CAN 等にもほぼ同様の仕組みを用いることが可能である．以下では Chord の用語に従って，Chord の ID リング上で時計回りの方向に隣接するホスト（群）を successor(s)，反時計回りの方向に隣接するホスト（群）を predecessor(s) とよぶことにする．Chord では stabilize プロトコルによってつねに successors を保持している．Pastry，CAN 等でも同様のプロトコルを導入することで successors に関する情報を保持することが可能である．

ここで用いるランダムなホスト選択の基本的な考え方は，ランダムな番号を生成し，その番号の successor をランダムホストとして選ぶというものである．ただし，このような方法ではホスト ID の分布が均一でないため，各ホストが選択される確率が均等にならない．具体的には各ホストが選択される確率は幾何分布に従うこととなる．したがって，平均的にみると最も選択されるホストは $\Theta(\frac{1}{N}\log N)$ の確率で選択されてしまうことになる．

そこで各ホストが選択される確率を平滑化するため，以下のような方法を採用する．先に述べたようにランダム ID の直近の successor を選ぶのではなく，ランダム ID からみて $s$ 個の successors を取り出し，その中からランダムに選んだものを選択することとする（**図 4**）．こうして選ぶことによって各ホストは

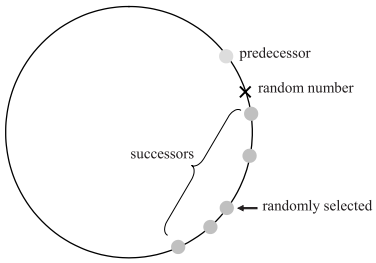


**図 4** ランダムホストの選択
Fig. 4 Random host selection.

$(1+\epsilon)\frac{1}{N}$ 以下の確率で選ばれることを確率的に証明できる（$s$ が $\Omega(\log N)$ のとき，$\epsilon$ は任意）．

ではあるホスト A からみて $s$ 番目の successor ホスト B が選ばれる確率 $p_b$ が $(1+\epsilon)\frac{1}{N}$ より大きくなる確率が低くできることを示す．ただし $I_a$，$I_b$ はホスト $A$，$B$ のホスト ID とし，$I_U$ は ID 空間の広さ $(2^{160})$ とする．ホスト ID は SHA-1 によって生成するものとする．

$$Pr\left[p_b > (1+\epsilon)\frac{1}{N}\right] \quad (1)$$

$$= Pr\left[\frac{I_b - I_a}{I_U \times s} > (1+\epsilon)\frac{1}{N}\right] \quad (2)$$

$$= Pr\left[(I_b - I_a) > (1+\epsilon)\frac{I_U \times s}{N}\right] \quad (3)$$

つまりホスト B が選ばれる確率 $p_b$ が $(1+\epsilon)\frac{1}{N}$ より大きくなる確率は ID 空間上のホスト B とホスト A の距離が $\frac{I_U \times s}{N}$ の $(1+\epsilon)$ 倍より大きくなる確率に等しい．この確率は $(I_a, I_a + (1+\epsilon)\frac{I_U \times s}{N}]$ の空間において存在するホストの数が $s$ 個より小さい確率に等しいので，以下ではこの空間におけるホストの数に関して考察を加える．

この空間上の点においてホストが存在するかどうかは各点独立であり，ベルヌーイ試行に従うと考えられる．したがって，この空間内に存在するホストの数は二項分布に従っており，$Bi((1+\epsilon)\frac{I_U \times s}{N}, \frac{N}{I_U})$ の確率変数 $X$ に関して $Pr[X < s]$ を求めればよい．ここで導き出したい結果は $s$ を $\Omega(\log N)$ にとったときに，この確率を $\frac{1}{N}$ にいくらでも近づけられるということである．

幸いにして二項分布に関するこの確率は Chernoff の不等式によって上限を与えられる．$X_1, X_2, ..., X_n$ が独立なベルヌーイ確率変数であり，$Pr[X_i] = p_i (0 < p_i < 1, 1 \leq i \leq n)$ のとき，$X = \sum_{i=1}^{n} X_i, \mu = E[X] = \sum_{i=1}^{n} p_i$ に関して以下の不等式が成り立つ $(0 < \delta \leq 1)$.

$$Pr[X < (1-\delta)\mu] < \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu} \quad (4)$$

$$Pr[X < (1-\delta)\mu] < e^{-\frac{\mu\delta^2}{2}} \quad (5)$$

上の式の方がより正確な上限を与えるが，ここでは簡単のため下の式を用いることとする．ここで $\delta = \frac{\epsilon}{1+\epsilon}$ となるように $\delta$ をとると，$1-\delta = \frac{1}{1+\epsilon}$ なので

$$Pr[X < s] = Pr\left[X < \frac{\mu}{1+\epsilon}\right] \quad (6)$$

$$= Pr[X < (1-\delta)\mu] \quad (7)$$

$$< e^{-\frac{\mu(\epsilon/1+\epsilon)^2}{2}} \quad (8)$$

ここで $\mu = (1+\epsilon)s$ を代入して，

$$Pr[X < s] < e^{-s\epsilon^2/2(1+\epsilon)} \quad (9)$$

したがって，$s = \left(2(1+\epsilon)/\epsilon^2\right)\ln N$ となるように，$s$ をとれば $Pr[X < s] < \frac{1}{N}$ となり，非常に高い確率でホスト B が選ばれる確率は $(1+\epsilon)\frac{1}{N}$ よりも低くなる．本節で述べた手法については 4.1 節で検証している．

### 3.6 ルックアッププロセス

本節では 3.1 節で触れたルックアップの流れについてより詳しく述べる．

**データの格納**

ルックアップの最初の過程は通常の分散ハッシュテーブルにおけるルックアップとほぼ同じである．ただし，キーはハッシュ関数によって生成するのではなく，(特徴量ベクトル等を）そのまま利用する．以上の過程でリダイレクション元の IP，ポート番号が得られる．

リダイレクション元は自分が管理するブロックのうちブロックサイズが小さいブロックを探し，そのブロックのリダイレクション先を返す．クライアントはリダイレクション先に直接データを送信し，データを格納する．

**データの検索**

クライアントは単一のキーもしくはキーの範囲を指定してリダイレクション元にリクエストを送る．リダイレクション元は自身が管理しているすべてのブロックのリダイレクション先のすべての IP，ポート番号をクライアントに返す．

クライアントはアプリケーションの要件に応じてリクエストを送信するリダイレクション先を選ぶ．該当するすべてのデータを取得したい場合には，すべてのリダイレクション先にリクエストを送ればよい．1 つのキーに対応するデータの数が多く，それぞれについて逐次検索を行わないとならないような場合，このような並列的な処理は好ましい．逆に 1 つのエントリのデータが少ない場合には無駄の多い方法であり，ブロックの構築方法を修正する必要があるが，煩雑なため本稿では扱わない．

以上が基本的な流れであるが，このような処理ではつねにリダイレクション元に対する通信が行われることになり，このホストに高い負荷がかかってしまうことになる．そこでキャッシングを利用することでこの問題を解決する．3.1 節で述べたとおりクライアントは Chord 等のルーティングを用いてリダイレクション元と通信を行っている．その際，Chord 等のルーティングの性質上，異なるクライアントから同じリダイレクション元へのリクエストは途中同じホストを経由することとなる．

したがって，あるクライアントが取得したリダイレクションポインタの情報をルーティング経路上のホストに短時間（100 ミリ秒から数秒）キャッシュしておけば，別のクライアントのリクエストはリダイレクション元の手前のホストで解決できることとなる．したがってリダイレクション元のホストへのリクエストはたかだか，数十ミリ秒程度の間隔でリクエストを処理すればよい．このようなルーティング経路上でのキャッシングは，分散ハッシュテーブルを用いたシステムにおいては一般的な技術である[5]．

## 4. 検　　証

本章では平均負荷についての知識を持たない場合にもブロックリダイレクションが効果的に動作するかについて調べた．また 2.4 節で列挙した要件を満たしているかについても検討した．

### 4.1 ランダムホストの選択に関する検証

3.5 節で述べたランダムホスト選択においてどの程度均一な確率でホストが選択されるかを確認した．

**設定条件**

1,000 個のホストに関してそれぞれ 32 ビットのランダムなホスト ID を生成した．そして 3.5 節の手順に従って $s$ 個の successors を取得し，その中からランダムにホストを選んでホストの選択結果とした．ランダムホストの選択は 1,000,000 回繰り返して行う．

**シミュレーション結果**

得られた結果を**図 5** に示す．横軸は候補として選ぶ successors の数 $s$，縦軸は各ホストが選ばれた確率を表している．グラフ中の縦線は分布を表している．縦線の上端は選択された回数の多いホスト，下端は選択された回数の少ないホストを示している．ただし上下それぞれ 1% の点は除去されている．また図中の Chernoff bound とは式 (9) の右辺の値を指している．

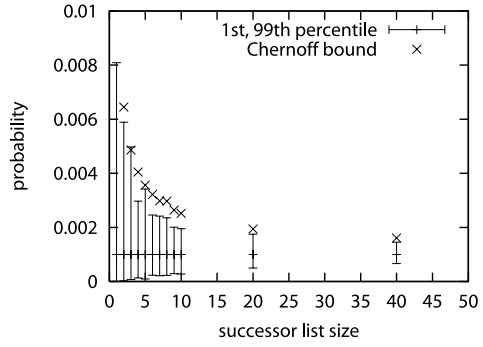


**図 5** 検証 1 の結果
Fig. 5 Verification 1: result.

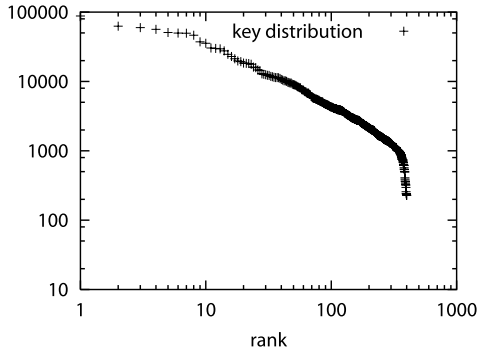


**図 6** 検証 2 に用いたデータの分布（$\alpha = 1.0$）
Fig. 6 Verification 2: data distribution.

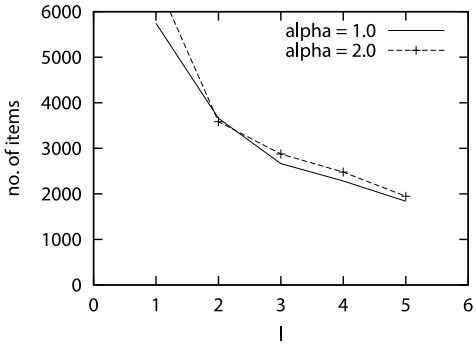


**図 7** 検証 2 の結果
Fig. 7 Verification 2: result.

図から明らかなようにChernoff boundとシミュレーションの結果はよく一致しているが，実際に各ホストが選ばれる確率は式 (9) で得られる確率より少し良いことがみてとれる．また図からくみとれるように，$s$ を大きくとるに従って各ホストが選択される確率が平均に近づく．

### 4.2 負荷分散特性に関する検証

ブロックリダイレクションによる負荷分散特性をシミュレーションによって確認した．

**設定条件**

まず 20 個の属性を用意し，それぞれについて 20 個のとりうる属性値を設ける．この 400 個の（属性，属性値）は Zipf 分布（指数 $\alpha$）に従って生成されるようにする．各オブジェクトにはこの分布に従って 20 個の（属性，属性値）をつけてゆく．ただし同一のオブジェクトにつける属性が重複した場合は再度選びなおす．このようにして 20 次元のキーを持つオブジェクトを 100,000 個生成し，前述のアルゴリズムに則って 100 個の CAN ホストに分散配置した．平均負荷の推定に用いる標本数は $m = 10$ である．ブロックサイズは標本平均とした．

**シミュレーション結果**

最終的な（属性，属性値）の分布を**図 6** に，シミュレーションの結果を**図 7** に記した．図 6 の横軸は（属性，属性値）の出現頻度の順位（rank）であり，縦軸は出現回数である．図から分かるとおり，キーは Zipf-like な非常に偏った分布を示している．

他方，図 7 の横軸はブロックのリダイレクト先の候補ホスト数 $l$，縦軸は最も負荷の高いホストに格納されるオブジェクト数である．Zipf 分布のパラメータ $\alpha$ が 1.0 と 2.0 のとき，それぞれについてシミュレーションを行ったが，ほぼ同じ結果が得られた．ともに偏りの強いキーの分布であるにもかかわらず，$l$ を増やすに従って最悪負荷が減少し，$l = 5$ のとき平均負荷（1,000）の 2 倍に近い値が得られていることが分かる．リダイレクト先に格納されるブロックの数がすべて等しくなる（すなわち 1 になる）確率はきわめて低いため，平均負荷との比率が 2 に近くなったのは我々の予想と一致する．完全に 2 にならないのは，ブロックサイズは均一ではないためであり，$l$ をこれ以上増やしても負荷分散の特性の改善はあまり見込めない．これ以上の改善を図るにはブロックサイズを小さくする必要がある．

### 4.3 要件に関する検証

本稿で述べたブロックリダイレクションが，2.4 節で列挙した要件を満たしているか検証する．

まず局所性の維持についてであるが，キーが同一のホストにマッピングされるものどうしでデータがブロック化されるため，局所性はほぼ維持される．同一のリダイレクション元にマッピングされたデータは必ずリダイレクション先のいずれかに存在する．

次に完全分散性に関してであるが，クエリごとの集中的なタスク分配は行われていないため，それに起因するクエリ処理のボトルネックが生じない．ブロックの管理という点に関してリダイレクション元によるタスクの調整は行われるが，この処理は頻繁に発生しな

いので一般的なデータ分布の偏りでは問題とはならない．

スケーラビリティに関しても，bins and balls という過程は必要な計算量が非常に少ないため，リダイレクションという処理自体はスケーラビリティに優れている．ただブロックサイズの調整を定期的に行う必要があり，データの分布が極端に偏っている場合には問題となりうる．ただブロックサイズは非常に大きく，一般には急激に変化しないので，頻繁に調整を行う必要はない．

そして要件の最後，最悪負荷の保証については，すでに述べているとおり平均負荷が既知の場合，平均負荷の $\Theta(\log\log N/\log l)$ 倍になるという保証が得られている．

## 5. 関連研究

我々の知る限り bins and balls 過程が初めて分散ハッシュテーブルに導入されたのは Byers らによる研究[3] においてである．ただ，彼らの手法では負荷分散をアイテム単位で行っており本稿のブロックリダイレクションのようにキーの局所性は維持されない．

本稿の手法以外で，キーの局所性を意識した負荷分散を行う手法としては Karger らによる手法[9] がある．この手法は ID 空間の分割に基づく方法であるため，単一のオブジェクトへの負荷が極端に重いような状況では対処できない．また，ホストの参加，離脱にともなう短期的なデータの移動コストが大きいというのも欠点の 1 つである．

仮想サーバ[10] を利用した方法もこの領域への適用が可能である．仮想サーバ方式の最大の利点は各ホストの処理容量の大きさに応じて負荷分散できる点にある．他方で，性能に関する保証がないこと，通信コストが大きい点が問題である．

そのほかにも汎用的な手法ではないが，pLSI における content-aware node bootstrapping も興味深い手法の 1 つといえる．これらの手法は基本的に ID 空間の分割に基づくものであり，単一オブジェクトにアクセスが集中するような場合については考慮されていない．

本稿で述べたブロックリダイレクションは特定の ID 空間に負荷が集中した場合，クエリの解決に必要なコストが増加してしまうという問題がある．したがって，リダイレクション元にマッピングされるデータ量を均一化するために ID 空間の分割に基づく方法を組み合わせるとより効果的である．このとき均一化には（キー，値）の移動をともなわないので，必要な通信量は少なくてすむ．

## 6. まとめ

本稿ではブロックリダイレクションという手法を導入し，キーの局所性を意識したスケーラブルな負荷分散の実現方法について述べた．示した手法は単純であるにもかかわらず，偏りの強いデータの分布についてもきわめて高い負荷分散の性能を得ることができた．クエリのアクセスパターンが動的に変化する環境下で適切に動作する負荷分散については今後の課題である．

## 参考文献


1) Azar, Y., Broder, A.Z., Karlin, A.R. and Upfal, E.: Balanced Allocations, *SIAM J. Comput.* (1999).
2) Bharambe, A.R., Agrawal, M. and Seshan, S.: Mercury: Supporting Scalable Multi-Attribute Range Queries, *Proc. Conference of the ACM Special Interest Group on Data Communication* (*SIGCOMM'04*) (2004).
3) Byers, J., Considine, J. and Mitzenmacher, M.: Simple Load Balancing for Distributed Hash Tables, *Proc. 2nd International Workshop on Peer-to-Peer Systems* (*IPTPS'03*) (2003).
4) Byers, J., Considine, J. and Mitzenmacher, M.: Geometric Generalizations of the Power of Two Choices, *Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures* (*SPAA'04*) (2004).
5) Dabek, F., Kaashoek, M.F., Karger, D., Morris, R. and Stoica, I.: Wide-area Cooperative Storage with CFS, *Proc. 18th ACM Symposium on Operating Systems Principles* (*SOSP'01*) (2001).
6) Datar, M., Indyk, P., Immorlica, N. and Mirrokni, V.: Locality-Sensitive Hashing Scheme Based on p-Stable Distributions, *Proc. 20th Annual Symposium on Computational Geometry* (*SCG'04*) (2004).
7) Deerwester, S., Dumais, S.T., Furnas, G.W., Laundauer, T.K. and Harshman, R.: Indexing by Lantent Semantic Analysis, *Journal of the American Society of Information Science* (1990).
8) Kaashoek, M.F. and Karger, D.R.: Koorde: A Simple Degree-optimal Distributed Hash Table, *Proc. 2nd International Workshop on Peer-to-Peer Systems* (*IPTPS'03*) (2003).
9) Karger, D.R. and Ruhl, M.: Simple, Efficient Load Balancing Algorithms for Peer-to-peer Systems, *Proc. 16th ACM Symposium on Parallelism in Algorithms and Architectures*

(*SPAA'04*) (2004).
10) Rao, A., Lakshminarayanan, K., Surana, S., Karp, R. and Stoica, I.: Load Balancing in Structured P2P Systems, *Proc. 2nd International Workshop on Peer-to-Peer Systems* (*IPTPS'03*) (2003).
11) Ratnasamy, S., Francis, P., Handley, M. Karp, R. and Shenker, S.: A Scalable Content-Addressable Network, *Proc. Conference of the ACM Special Interest Group on Data Communication* (*SIGCOMM'01*) (2001).
12) Rowstron, A. and Druschel, P.: Pastry : Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems, *Proc. IFIP/ACM International Conference on Distributed Systems Platforms* (*Middleware*) (2001).
13) Stoica, I., Morris, R., Kaashoek, M.F. and Balakrishnan, H.: Chord : A Scalable Peer-to-Peer Lookup Service for Internet Applications, *Proc. Conference of the ACM Special Interest Group on Data Communication* (*SIGCOMM'01*) (2001).
14) Tang, C., Xu, Z. and Mahalingam, M: pSearch: Information Retrieval in Structured Overlays, *ACM SIGCOMM Computer Communication Review* (2003).
15) Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. and Kubiatowicz, J.D.: Tapestry: A Resilient Global-Scale Overlay for Service Deployment, *IEEE Journal on Selected Areas in Communications* (2004).

(平成 17 年 5 月 19 日受付)
(平成 17 年 11 月 1 日採録)

**岡　敏生**（学生会員）

平成 13 年京都大学工学部情報学科卒業．平成 15 年東京大学大学院新領域創成科学研究科基盤情報学専攻修士課程修了．現在，同博士課程在学中．分散コンピューティング，確率的アルゴリズムなどの研究に従事．電子情報通信学会会員．

**森川　博之**（正会員）

昭和 62 年東京大学工学部電子工学科卒業．平成 4 年同大学大学院工学系研究科電気工学専攻博士課程修了．平成 9 年から平成 10 年にかけてコロンビア大学客員研究員．現在，東京大学大学院新領域創成科学研究科基盤情報学専攻助教授．工学博士．コンピュータネットワーク，ユビキタスネットワーク，モバイルインターネット，分散コンピューティングなどの研究に従事．丹羽記念賞，電子情報通信学会篠原記念学術奨励賞，電子情報通信学会論文賞等受賞．IEEE，ACM，ISOC，電子情報通信学会，映像情報メディア学会各会員．

**青山　友紀**（正会員）

昭和 42 年東京大学工学部電子工学科卒業．昭和 44 年同大学大学院工学系研究科電気工学科修士課程修了．同年日本電信電話公社（現 NTT）入社．以来，電気通信研究所において，伝送システム，ディジタル信号処理，広帯域ネットワークなどの研究に従事．平成 9 年に東京大学に転じ，現在，同大学大学院情報理工学系研究科電子情報学専攻教授．工学博士．電子情報通信学会副会長．電子情報通信学会通信ソサイエティ編集長．IEEE Fellow．超高速フォトニックネットワーク開発推進協議会会長．ディジタルシネマコンソーシアム会長．ユビキタスネットワークフォーラム副会長．