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

2004－DBS－134　(II)　(92)
2004／7／15

P2P ネットワークにおける一般化 Kautz ダイグラフに基づく分散ハッシュ表を用いた検索アルゴリズム

岡下 綾[*a]　　有次 正義[*b]　　柴田 幸夫[*b]

{aya,shibata}@msc.cs.gunma-u.ac.jp, aritsugi@dbms.cs.gunma-u.ac.jp

概要

P2P(Peer-to-Peer) ネットワークではデータを動的なネットワーク上のノードに分散させるため，データを効率良く検索することが本質的に重要な課題となる．そこで，ネットワーク上の各ノードに経路情報として分散ハッシュ表 (DHT) を持たせることにより，位置に依存せず効率的にデータを検索するアプローチが注目を集めている．本稿では一般化 Kautz ダイグラフに基づく定数サイズの DHT を提案する．各ノードは他の二つのノードに関する情報を持つだけで，Koorde よりも拡張性の高いネットワークの上で，ノード数 n に対してホップ数 $O(\log n)$ の検索を実現する．

キーワード：P2P ネットワーク，分散ハッシュ表，一般化 Kautz ダイグラフ，分散データ発見アルゴリズム．

On Lookup Algorithm with a Generalized Kautz-based Distributed Hash Table

AYA OKASHITA[*a] MASAYOSHI ARITSUGI[*b] and YUKIO SHIBATA[*b]

{aya,shibata}@msc.cs.gunma-u.ac.jp, {aritsugi}@dbms.cs.gunma-u.ac.jp

Abstract

In peer-to-peer(P2P) networks, it is important to efficiently locate nodes assigned to data items. This paper proposes lookup algorithm with a distributed hash table(DHT) based on generalized Kautz digraphs as a routing table. This is a family of constant-degree routing networks of logarithmic diameter. Generalized Kautz digraphs have an optimal diameter and higher scalability than de Bruijn digraphs. In our method, each node needs to store routing information about only 2 other nodes. Our algorithm guarantees $O(\log n)$ hops, for the number of nodes n.

Keywords: Peer-to-Peer Network, Distributed Hash Table, Generalized Kautz Digraph, Lookup Algorithm.

1　はじめに

インターネットの大部分はサーバベースサービスであり，すべての通信は中央サーバを介して行われる．そのため，クライアント端末（ノード）は中央サーバから集中的な管理や監視，制御を受ける．

Peer-to-Peer (P2P) ネットワークはノード同士が対等に通信を行う自律分散ネットワークである．P2P ネットワークは高価な中央サーバを必要とせず，容易に柔軟で耐故障性に優れたネットワークを構築できる．ノードは自由に参加／離脱を行い，ネットワークトポロジーは動的に変化する．すべてのノードに対等の権限が与えられ，ネットワークに接続されているすべてのノードが持つデータにアクセスすることが可能となる．通信は個々のノード間で直接行われるため，プライバシーの保護も容易である．

[*a] 群馬大学大学院電子情報工学専攻
Department of Computer Science, Postgraduate School of Gunma University
[*b] 群馬大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Gunma University

現在，P2P ネットワークはファイル共有や分散ストレージなどに多く利用されている．P2P を利用したファイル共有システムに Gnutella[1]，Napster[2] などが挙げられる．P2P ネットワークにおいて，各ノードは隣接するノードの情報のみを持ち，ネットワーク全体の情報を持つノードは存在しない．そのため，要求されたデータを持つノードを検索する問題は本質的に重要である．データを検索するためには，隣接するノードに問合せ（クエリ）をリレーで渡していく方法を採る．

Gnutella を代表とするブロードキャスト型ファイル共有システムでは，隣接するすべてのノードに対してクエリを送信する．ノード数が増えると，検索の際に生じるトラフィックやメッセージ数は膨大となるため，検索範囲を大幅に制限する必要がある．そのため，ノードの位置によって検索できないデータが存在してしまう．

そこで，ネットワーク上の各ノードに経路情報として分散ハッシュ表 (DHT) を持たせることにより，位置に依存せず効率的にデータを検索するアプローチが注目を集めている [7]．DHT を用いる検索は，オーバーレイネットワーク上の検索となる．オーバーレイネットワークに様々なダイグラフのサブクラスを埋め込むことにより，ダイグラフのルーティングアルゴリズムを利用して，確実かつ効率的に検索を行うことができる．

DHT の主な評価指標に，次数とホップ数がある．次数は各ノードに隣接するノードの数であり，ホップ数はデータを検索するために必要な仲介ノードの数の期待値である．次数とホップ数には一般にトレードオフの関係がある．これまでに Chord[8]，Pastry[4]，Tapestry[5]，Viceroy[6]，HyperCup[11]，D2B[13]，Koorde[10] などの DHT プロトコルが提案されている．これらの DHT アルゴリズムでは，ノード数 n に対して $O(\log n)$ のホップ数を実現している．

本稿では，一般化 Kautz ダイグラフ [14] に基づく DHT を用いた分散検索アルゴリズムを提案する．これは Chord DHT プロトコルの族に含まれ，類似の検索アルゴリズムとして Koorde が存在する．Chord は ring と finger table に基づいた DHT を用いることにより，次数 $\log n$ に対して $O(\log n)$ のホップ数を実現している．Koorde は Chord ring に de Bruijn ダイグラフ [12] を埋め込むことにより，各ノードはわずか二つの他のノードの情報を保持するだけで $O(\log n)$ のホップ数を実現している．de Bruijn ダイグラフ，Kautz ダイグラフ [16] は次数とノード数に対して最小の直径を持つダイグラフであり，Moore bound[15] に対して良い値を持つことが知られている．直径とは，最も遠いノード間の距離であり，直径が小さいほど性質の良いネットワークトポロジであるといえる．検索アルゴリズムのホップ数はネットワークの直径の値に大きく依存する．Kautz 族のダイグラフは de Bruijn 族のダイグラフよりも拡張性の高いダイグラフである．直径 D，次数 d に対して，de Bruijn ダイグラフが d^D 個のノードを持つのに対して，Kautz ダイグラフは $d^D + d^{D-1}$ 個のノードを持ち，より拡張性が高いことがいえる．本稿で取り扱う一般化 Kautz ダイグラフ [14] は Kautz ダイグラフを代数的な構成法から一般化したダイグラフであり，Kautz ダイグラフの性質を失わないまま，任意のノード数に対応している．一般化 Kautz ダイグラフはノード数に対して最小かそれに近い直径を持つ．一般化 Kautz ダイグラフは Kautz ダイグラフを含んだ，より大きいダイグラフのクラスであることから，本稿で提案するアルゴリズムは Kautz ダイグラフに対しても適用できる．

本稿の構成を以下に示す．2 節では，一般化 Kautz ダイグラフの定義とルーティングアルゴリズムについて述べる．3 節では，一般化 Kautz ダイグラフに基づく分散ハッシュ表と提案したネットワークトポロジにおけるデータ検索アルゴリズム，さらにアルゴリズムの高速化やネットワーク管理コストなどについて述べる．最後に 4 節において，まとめを述べる．

2 一般化 Kautz ダイグラフとルーティングアルゴリズム

2.1 一般化 Kautz ダイグラフ

次数 d，ノード数 n の一般化 Kautz ダイグラフ $G_K(d, n)$ のノード集合を V，弧集合を A で表す．一般化 Kautz ダイグラフは $V = \{0, 1, \ldots, n-1\}$，$A = \{(x, y) \mid y \equiv -dx - i \pmod{n},\ 1 \leq i \leq d\}$ として定義される [9, 14]．

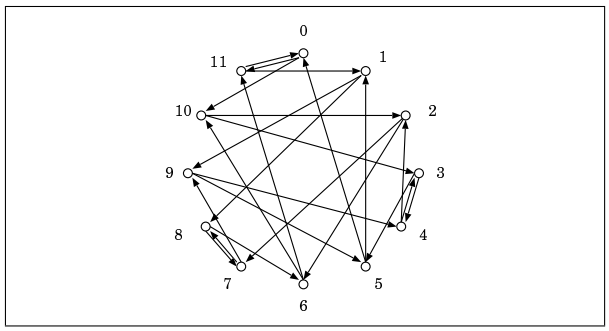

図 1: 一般化 Kautz ダイグラフ $G_K(2,12)$

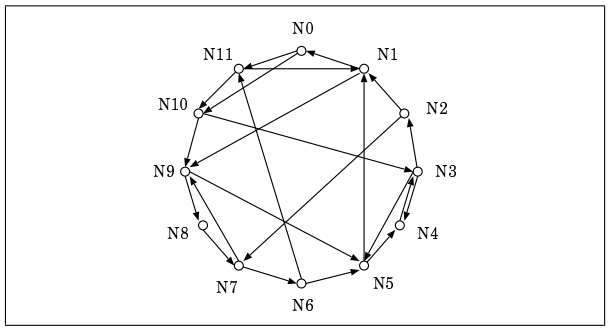

図 2: 一般化 Kautz ダイグラフに基づく分散ハッシュ表

直径は $\lfloor \log_d n \rfloor$ 以上 $\lceil \log_d n \rceil$ 以下であり，ノード数に対して最小かほぼ最小となる [15]．一般化 Kautz ダイグラフは $n = d^D + d^{D-1}$ ならば Kautz ダイグラフと同型であり，$n = d^D$ ならば一般化 de Bruijn ダイグラフや de Bruijn ダイグラフと同型である．一般化 Kautz ダイグラフは Kautz ダイグラフを含み，一般化 de Bruijn ダイグラフと共通集合を持つダイグラフのサブクラスである．

図 1 に次数 2，ノード数 12 の一般化 Kautz ダイグラフ $G_K(2,12)$ を示す．

2.2 一般化 Kautz ダイグラフにおけるルーティング

ダイグラフ G において，ノード列 $u_0, u_1, \ldots, u_\ell$ に対して，弧 (u_i,u_{i+1})，$0 \le i \le \ell - 1$，が存在するならば $u_0, u_1, \ldots, u_\ell$ を G における長さ ℓ の (u_0,u_ℓ) ウォーク，または単にウォークと呼ぶ．

次数 d，ノード数 n の一般化 Kautz ダイグラフ $G_K(d,n)$ におけるノード x から y への経路は，長さが最小のウォークを特定することにより得られる．ノード x から y への長さ ℓ のウォークを $x = u_0, u_1, u_2, \ldots, u_\ell = y$ とすると，合同式 (1) を得る．合同式はすべて n を法とする．

$$\begin{cases} u_1 \equiv -du_0 - r_0 \\ u_2 \equiv -du_1 - r_1 \\ u_3 \equiv -du_2 - r_2 \\ \vdots \\ u_\ell \equiv -du_{\ell-1} - r_{\ell-1} \end{cases} \quad (1)$$

ノード x から y への最短経路を求めることは，合同式 (1) を満足する最小の ℓ の値と $r_0, r_1, \ldots, r_{\ell-1}$，$r_i \in \{1, 2, \ldots, d\}$，$0 \le i \le \ell - 1$ の値を求めることと等価である．一般化 Kautz ダイグラフにおけるノード x からノード y への長さ ℓ の経路が存在するための必要十分条件を補題 2.1 に示す．

補題 2.1 合同式 (1) が解を持つための必要十分条件は以下の条件を満たす整数 t が存在することである．

1. ℓ が奇数ならば，
$$1 \le t \le d^\ell, \quad t \equiv (-d)^\ell x - y \pmod{n}$$
2. ℓ が偶数ならば，
$$-d^\ell + 1 \le t \le 0, \; t \equiv (-d)^\ell x - y \pmod{n}$$

■

$1 \le \ell \le \log n$ に対して，$t \equiv (-d)^\ell x - y \pmod{n}$ を求めることにより，条件を満たす最小の ℓ の値を見つけることができる．得られた最小の ℓ に対する r_i，$0 \le i \le \ell - 1$，は，長さ ℓ のウォークに一意に対応する．得られるウォークの長さは最短なので，同じノードを重複して通らない．

補題 2.1 より，r_i，$0 \le i \le \ell - 1$ の値は次のように求めることができる．ℓ が奇数の場合，$t \equiv (-d)^\ell x - y - 1 \pmod{n}$，$0 \le t \le d^\ell - 1$ とし，$t_0 t_1 \ldots t_{\ell-2} t_{\ell-1}$ を t の d 進表記とすると，i が偶数ならば $r_i = t_i + 1$，i が奇数ならば $r_i = d - t_i$ である．ℓ が偶数の場合，$t \equiv d^\ell(x+1) - (y+1) \pmod{n}$，$0 \le t \le d^\ell - 1$ とし，$t_0 t_1 \ldots t_{\ell-2} t_{\ell-1}$ を t の d 進表記とすると，i が偶数ならば $r_i = d - t_i$，i が奇数ならば $r_i = 1 + t_i$ である．

例として，一般化 Kautz ダイグラフ $G_K(2,12)$ における (2,4) ウォークを求める（図 1）．$\ell = 1, 2$ に対しては，補題 2.1 の条件は満たされない．$\ell = 3$ に対して，$t \equiv (-2)^3 \cdot 2 - 4 \pmod{12} \equiv 4$ より，$1 \le t \le 2^3$ という条件を満たし，長さ 3 のウォークが存在することが分かる．$(-2)^3 \cdot 2 - 4 - 1 \equiv 3$

(mod 12) より，3 を 2 進表記すると 011 なので $r_0 = 1+0 = 1$，$r_1 = 2-1 = 1$，$r_2 = 1+1 = 2$ より，ウォーク $2, 7, 9, 4$ が得られる．

3 分散ハッシュ表とデータ検索アルゴリズム

3.1 分散ハッシュ表

ネットワーク上の各ノードとデータには SHA-1[3] などのハッシュ関数を用いて 0 から $N-1$ までのハッシュ値を識別子としてそれぞれに割り当てる．ノードの識別子は端末の IP アドレスやポート番号などから，データの識別子はファイル名やファイル情報などからハッシュ値として得る．ノードに割り当てられる識別子を NID，データに割り当てられる識別子を KID と表記する．NID には N を，KID には K を識別子の頭に付けることにより双方を区別する．

N 個の識別子で構成される論理的な識別子円に，割り当てられている参加ノードの数を n とする．各識別子 id に対して，識別子円上で反時計回りに先に存在する最初のノードと，その NID を id の *forward* と呼び，$forward(id)$ で表す．また，id に対して，識別子円上で反時計回りに id 自身か id より先に存在する最初のノードと，その NID を id の *successor* と呼び，$successor(id)$ で表す．また id に対して，識別子円上で反時計回りに id より手前に存在する最初のノードと，その NID を id の *predecessor* と呼び，$predecessor(id)$ で表す．KID id を持つデータは，$successor(id)$ を識別子として持つノードに割り当てられる．本稿を通じて，識別子に関する計算はすべて N を法とし，識別子の取り得る範囲は 0 以上 $N-1$ 以下とする．また，$t = x \pmod N$ と表記された場合，$0 \le t < N$ であることを仮定する．識別子空間において，反時計回りの向きで連続する識別子の集合を区間と呼び，閉区間表記 $[,]$ と開区間表記 $(,)$ を用いて表す．識別子 x から y までの区間は $[x, y] = \{x, x+1, x+2, \ldots, y-1, y\}$ となる．$(x-1, y]$，$[x, y+1)$，$(x-1, y+1)$ も $[x, y]$ と同じ区間を表す．区間 $[x, y]$ に含まれる識別子の数を $|[x, y]|$ で表す．

すべての識別子に対してノードが割り当てられていれば検索経路は最短となるため，検索に必要なホップ数は，検索を実行したノードとデータを保持するノードの間の距離に等しくなる．しかし，ネットワークに多くのノードがいつでも参加できるように，またハッシュ値の衝突を防ぐためにも，識別子円の状態は疎にしておかなければならない．よって N は十分大きい値を取り，ノードが割り当てられない多くの NID（仮想ノード）が存在する．

KID k を持つデータを検索するために，一般化 Kautz ダイグラフの弧を辿ることにより successor(k) を発見する．識別子円に埋め込まれた一般化 Kautz ダイグラフは多くの仮想ノードを含むため，理論上の最短経路に含まれた仮想ノードを辿ることはできない．そこで，仮想ノードの NID x に対する $predecessor(x)$ を辿ることで，ルーティングを行う．

$forward$ ポインタのみを辿ることにより検索中のデータに確実に到達できるが，最悪の場合にはすべてのノードを辿る必要がある．よって，$forward$ ポインタで構成される識別子円上に次数 2 の一般化 Kautz ダイグラフの弧を埋め込むことにより，$O(\log n)$ のホップ数を実現する．本稿で提案する DHT では，識別子 x を持つノードに次の二つのノードへのポインタ（IP アドレスやポート番号などの情報）を持たせる．

- $forward(x)$,
- $predecessor(-2x-1)$.

次数 2 の一般化 Kautz ダイグラフにおいて，ノード x は二つのノード $-2x-1$ と $-2x-2$ への弧を持つため，$predecessor(-2x-1)$ と $predecessor(-2x-2)$ の両方へのポインタが必要であるが，$predecessor(-2x-2)$=$predecessor(-2x-1)$ である場合が多く，それ以外の場合には $predecessor(-2x-2)$=$-2x-2$ であるので，$predecessor(-2x-1)$ のみを持たせる．$predecessor(-2x-1)$ から $forward$ ポインタを辿ることで $predecessor(-2x-2)$ に到達できる．

図 2 に仮想ノードを含まない状態の，一般化 Kautz ダイグラフに基づいた分散ハッシュ表を示す。

Require: x: NID, k: KID
Ensure: tp: 次の *turning point* の NID
  for $\ell = 2$ to D **do**
    if ℓ is even **then**
      $t \Leftarrow 2^\ell x - k + 2^\ell - 1 \pmod N$
      if $0 \le t \le 2^\ell - 1$ **then**
        $t_0 \Leftarrow t$ の最左ビット
        $r \Leftarrow 2 - t_0$
        return($-2x - r \pmod N$)
      end if
    else {ℓ is odd}
      $t \Leftarrow -2^\ell x - k - 1 \pmod N$
      if $0 \le t \le 2^\ell - 1$ **then**
        $t_0 \Leftarrow t$ の最左ビット
        $r \Leftarrow t_0 + 1$
        return($-2x - r \pmod N$)
      end if
    end if
  end for

図 3: 関数: find_route

Require: k: KID, p: 次の *turning point* の NID
Ensure: $successor(k)$: NID
  x : Lookup を実行するノードの NID
  if $k \in (x, forward]$ **then**
    return($forward$)
  else if $p == x$ **then**
    // x が *turning point* である場合
    $p \Leftarrow$ find_route(x,k)
    return($predecessor(-2x-1)$.Lookup(k,p))
  else if $p \in (x, forward]$ **then**
    $p \Leftarrow$ find_route(p,k)
    return($forward$.Lookup(k,p))
  else if $predecessor(-2x-1) \notin (x, p]$ **then**
    // $predecessor(-2x-1)$ が x よりも p から遠い場合
    return($forward$.Lookup(k,p))
  else
    return($predecessor(-2x-1)$.Lookup(k,p))
  end if

図 4: アルゴリズム: Lookup

3.2 経路計算

関数 find_route(x: NID, k: KID) は，一般化 Kautz ダイグラフにおけるノード x から k までの仮想ノードの存在を考慮しない理論上の最短経路を計算し，ノード x が Lookup を送信するノードの識別子を返す．図 3 に，関数 find_route を示す．ノード列 $u_0, u_1, \ldots, u_\ell$ が KID k を検索するための理論上の経路であるとき，各ノード u_i を *turning point* と呼ぶ．

3.3 検索アルゴリズム

図 4 に検索アルゴリズム Lookup を示す．
アルゴリズム Lookup(k: KID, p: NID) において，引数 k は検索データの識別子であり，p は経路上の次の *turning point* の識別子を表す．

アルゴリズム Lookup に必要なホップ数に関する補題を以下に示す．

補題 3.1 アルゴリズム Lookup に必要なホップ数の期待値は $3\lceil \log N \rceil$ である．

証明． まず，データ検索の流れを説明する．

検索を開始するノード i が KID k を検索するとき，最初に自分自身がデータを持っていないことを確認し，$predecessor(-2i-1)$.Lookup(k,find_route(i,k)) を実行する．

検索の過程で，Lookup(k,p) を渡されたノード x は次の処理を行う．ノード x の $forward$ が $successor(k)$ であれば，$forward$ を返す．ノード x が *turning point* ならば，次の *turning point* を計算して，$predecessor(-2x-1)$ に Lookup(k,p) を渡す．*turning point* p が $forward$ であるか，または $x = predecessor(p)$ ならば，次の *turning point* の NID を p として，$forward$ に Lookup(k,p) を渡す．

以上の条件を満たさない場合は，仮想ノード x からノード p へのルーティングのシミュレートに対応した処理を行う．$s = predecessor(x)$, $s' = forward(s)$, $p = predecessor(-2x-2) = -2x-2 \pmod N$ とし，s から p に到達するために必要なホッ

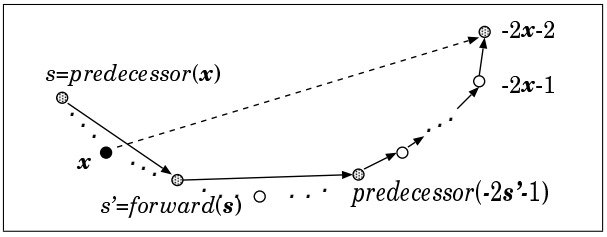

図 5: 仮想ノード x から $predecessor(-2x-2)$ へのルーティング

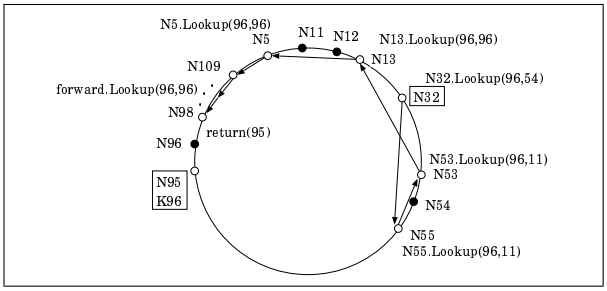

図 6: 識別子円における実際の経路の例

プ数について考察する．$predecessor(-2s'-1)$ が区間 $(x,p]$ に存在しない場合, つまり, 仮想ノード x から s' までの距離がある程度大きい場合には，$forward$ ポインタを辿って Lookup を渡すことにより p に到達する．そうでない場合には, $predecessor(-2x-1)$ へ Lookup(k,p) を渡す.

Lookup アルゴリズムに必要なホップ数を求めるために，*turning point* に仮想ノードが含まれ，最もホップ数が大きくなる場合を考察する．図 5 に仮想ノード x からノード $predecessor(-2x-2)$ へのルーティングをシミュレートする実際の Lookup クエリの流れを示す．点線は理論上の経路であり，実線は実際に辿る経路を表す.

図 5 において，$s = predecessor(x)$ から $p = predecessor(-2x-2)$ に到達するために，s はまず $s' = forward(s)$ に Lookup(k,p) を渡す．次に，s' は $predecessor(-2s'-1)$ に Lookup(k,p) を渡す．一般化 Kautz ダイグラフの定義より，$predecessor(-2s'-1)$ が区間 $(s',p]$ に存在するための必要十分条件は，$|(s',p]| > 2|(x,s']|$ であることである．よって，$predecessor(-2s'-1)$ から p に到達するまでに $forward$ ポインタを辿る回数 $|(predecessor(-2s'-1),p]|$ の値は高々 $u = (-2s'-1-(-2x-2)) \pmod N = 2(x-s')+1 \pmod N$ となる．N 個の識別子に対して n 個のノードが識別子円上にランダムに配置されているため，$|(s,s')|$ の期待値は $N/n-1$ であり，$|(x,s')|$ の期待値は $n(\sum_{i=0}^{N/n-1} i)/N = (N/n-1)/2$ となる．$x-s' \pmod N$ の期待値は $|(x,s')|+2$ であることから，$u = 2(|(x,s')|+2)+1 = N/n+1$ がいえる.

よって, $|(predecessor(-2s'-1), predecessor(-2x-2))|$ の期待値は $un/N = (N/n+1)n/N \doteq 1$ となる.

故に, 仮想ノード x からノード $predecessor(-2x-2)$ に到達するために必要なホップ数の期待値は，ノード $s = predecessor(x)$ から $s' = forward(s)$ へ 1 回，ノード s' から $predecessor(-2s'-1)$ へ 1 回，$predecessor(-2s'-1)$ から $predecessor(-2x-2)$ へのホップ数の期待値は 1 回なので 3 となる.

そうでない場合，つまり $|(s',p]| \leq 2|(x,s']|$ である場合には，s' は p に到達するために，$forward$ ポインタのみを辿る．このとき $forward$ ポインタを辿る回数は区間 $(s',p]$ に存在するノードの数である．$|(s',p]| \leq 2|(x,s']|$ かつ $|(x,s')|$ の期待値が $(N/n-1)/2$ であることから，区間 $(s',p]$ に存在する識別子の数の期待値は $N/n+1$ となる.

よって，$forward$ ポインタを辿る回数の期待値は $(N/n+1)\cdot n/N = 1+n/N \doteq 1$ となる（N は十分大きい値を取る）．故に，ノード s からノード p に到達するために必要なホップ数の期待値はノード $s = predecessor(x)$ から $s' = forward(s)$ へ 1 回，s' から p までのホップ数の期待値は 1 回なので 2 となる.

よって，アルゴリズム Lookup に必要なホップ数の期待値は $3\lceil \log N \rceil$ となる．■

補題 3.1 より，次の系が得られる.

系 3.2 アルゴリズム Lookup に必要なホップ数は $O(\log N)$ である．■

$N = 120$ に対してノード $N32$ がデータ $K96$ を検索する例を用いて具体的な説明を行う．検索で Lookup が渡される実際の経路を図 6 に，理論上の経路を図 7 に示す．図 6, 7 において，白い点は実ノードを，黒い点は仮想ノードを表す．始めに，ノード $N32$ は自分がデータ $K96$ を保持して

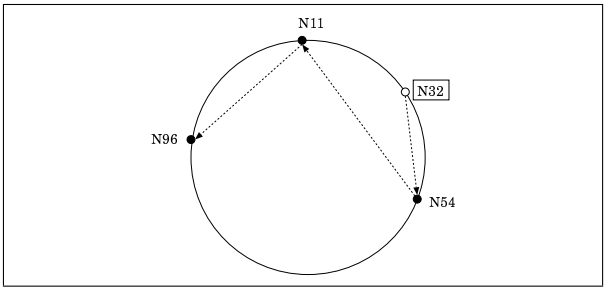

図 7: 識別子円における理論上の経路の例

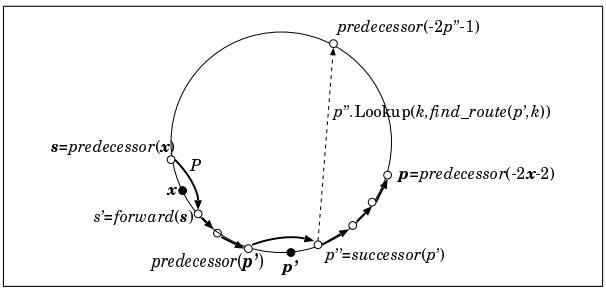

図 8: 経路変更の様子

いないことを確認して，次の *turning point* $N54$ を計算する．$N54$ は仮想ノードであるため $N32$ は $predecessor(N54) = N55$ に Lookup($K96$,$N54$) を渡す．$N55$ は $predecessor(N54)$ なので，次の *turning point* $N11$ を計算し，$forward(N55) = N53$ に Lookup($K96$,$N11$) を渡す．$predecessor(-2 \cdot 53 - 1) = N13$ は区間 $(N53, N11]$ に存在するため，$N53$ は次の *turning point* $N96$ を計算して，$N13$ に Lookup($K96$,$N96$) を渡す．$N13 = predecessor(N11)$ であるため，$N13$ は $forward(N13) = N5$ に Lookup($K96$,$N96$) を渡す．$predecessor(-2 \cdot 5 - 1) = N109$ は区間 $(N5, N96]$ に存在するため，$N5$ は $N109$ に Lookup($K96, N96$) を渡す．$N109$ から $predecessor(N96)$ に到達するまで，$forward$ に対して Lookup($K96$,$N96$) を渡していく．最後に，$N98 = predecessor(N96)$ が $N95 = forward(N98)$ を返す．

3.4 アルゴリズムの高速化

アルゴリズム Lookup は $O(\log N)$ のホップ数を必要とするが，N は非常に大きい数 (Chord では 2^{150}) であることが想定される．よって本節では，$O(\log n)$ のホップ数でデータを検索するための手法について考察する．

一般化 Kautz ダイグラフにおけるルーティングは，遠くのノードに飛んで，また近くのノードに戻ってくることにより，少しずつ目的のノードに近づいていくという規則的な軌道を持つ．通常，N の値は非常に大きい値を取り，仮想ノードの数も多くなることから，仮想ノード x からある *turning point* p への移動中に，後で辿るべき別の *turning point* p' に対する $predecessor(p')$ に到達してしまう場合も少なくない．そのような場合には，p ではなく p' からの経路を再計算して辿ることによりホップ数を $O(\log N)$ から $O(\log n)$ にまで削減できる．

turning point x を仮想ノードとし，$s = predecessor(x)$, $s' = forward(s)$ とする．区間 (s, s') に含まれる任意の仮想ノード i に対して，$s = predecessor(i)$, $s' = successor(i)$ なので，経路上の *turning point* を x から i に変更できる．

仮想ノード x からノード p (p は次の *turning point*) へのホップに対応する経路 P を図 8 に示す．図 8 において，白い点は実ノード，黒い点は仮想ノードを表す．

経路 P に含まれるノードで，理論的な経路上で最も k に近い *turning point* p' が存在する場合を考える．p' の存在は関数 $find\_route$ を少し変更することで，簡単に調べることができる．$predecessor(p')$ に到達したとき，$forward$.Lookup(k,find_route(p',k)) を実行することで，ホップ数を削減できる．

補題 3.1 において，ノード s から区間 $[predecessor(-2s' - 1), predecessor(-2s - 1)]$ に存在する任意のノード y に対して定数ホップで移動できることを示した．一般化 Kautz ダイグラフの定義より，区間 $[predecessor(-2s' - 1), predecessor(-2s-1)]$ に含まれる識別子の数は区間 $[s, s']$ に含まれる識別子の数 i に対して $2i-1$ となる．また，区間 $[predecessor(2 \cdot predecessor(-2s' - 1) - 1), predecessor(2 \cdot predecessor(-2s - 1) - 1)]$ に存在する任意のノード z に対して，定数ホップで到達できるような識別子 y に対する $predecessor(y) \in [predecessor(-2s' - 1), predecessor(-2s - 1)]$ が存在する．このように 1 回の定数ホップによって，

ノード s から到達できる識別子の数は少なくとも 2 倍 -1 ずつ増えていく．よって，ノード s から ℓ 回の定数ホップで到達できる識別子の数は少なくとも $2^{\ell}|[s,s']| - (2^{\ell}-1)$ となる．$|[s,s']|$ の期待値は $N/n+1$ なので，$2^{\ell}(N/n+1) - (2^{\ell}-1) > N$ となる最小の ℓ の値を求めると $\ell = \lceil \log n \rceil$ となる．

よって最適な *turning point* p' を選ぶことで，検索に必要なホップ数は $O(\log n)$ となる．

3.5 管理コスト

本稿で提案した DHT は *forward* ポインタを持つため，*forward* ポインタのみを辿るだけで，必ず *turning point* に到達できる．よってノードの参加／離脱，安定化に対して，Chord プロトコルで扱うアルゴリズムを適用できることから，Chord 同様の参加／離脱コストと安定性を持つ．

4 まとめ

本稿では，一般化 Kautz ダイグラフに基づく定数サイズの分散ハッシュ表と，ノード数 n に対して $O(\log n)$ のホップ数でデータを検索するアルゴリズムを提案した．今後の課題としては，分散ハッシュ表のサイズに関する一般化を行い，耐故障性について考察することなどが挙げられる．

参考文献

[1] Gnutella. http://gnutella.wego.com/

[2] Napster. http://www.napster.com/

[3] FIPS 190-1, “Secure Hash Standard,” *U.S. Department of Commerce/NIST, National Technical Information Service*, Springfield, VA, Apr. 1995.

[4] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pp.329-350, Nov., 2001.

[5] B. Zhao, J. Kubiatowicz and A. Joseph, “Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing,” *Technical Report UCB/CSD-01-1141*, U.C.Berkeley, CA, 2001.

[6] D. Malkhi, M. Naor, D. Ratajczak, “Viceroy: A Scalable and Dynamic Emulation of the Butterfly,” *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[7] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, I. Stoica, “Looking up data in P2P systems, ” *Communications of the ACM*, Vol.46, No.2, pp.43-48, Feb., 2003.

[8] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,” *IEEE/ACM Transactions on Networking*, Vol.11, No.1, pp.17-32, 2003.

[9] M. Imase and M. Ito, “A design for directed graphs with minimum diameter,” *IEEE Trans. Computer*, C-32, pp.782-784, 1983.

[10] M. Kaashoek and D. R. Karger, “Koorde: A simple degree-optimal distributed hash table,” *In Proc. of the 2nd International Workshop on Peer-to-Peer Systems(IPTPS)*, Mar., 2003.

[11] M. Schlosser, M. Sintek, S. Decker and W. Nejdl, “HyperCuP Hypercubes, Ontologies and Efficient Search on P2P Networks,” *In International Workshop on Agents and Peer-to-Peer Computing*, Jul., 2002.

[12] N. de Bruijn, “A combinatorial problem,” *Proc. Kon. Nederl. Akad. Wetensch*, Ser. A, 49, pp.758-764, 1946.

[13] P. Fraigniaud and P. Gauron, “Brief Announcement: An Overview of the Content-Addressable Network D2B,” *Brief Announcement at 22nd ACM Symp. on Principles of Distributed Computing (PODC)*, Jul., 2003.

[14] S. Reddy, D. Pradhan and J. Kurl, “Direct graphs with minimum diameter and maximal connectivity,” *School of Engineering, Oakland University Tech. rep.*, Jul., 1980.

[15] W. Bridges and S. Toueg, “On the impossibility of directed Moore graphs,” *J.Combinatorial Theory*, Ser. B, 29, pp.339-341, 1980.

[16] W. Kautz, “Bounds on directed (d,k)graphs,” *Theory of cellular logic networks and machines*, SRI Project 7258, AFCRL 68-0668 Final Report, pp.20-28, 1968.