＜報文＞

木構造型分散ハッシュテーブルを用いた P2P システム Kademlia の実装と評価

沖縄職業能力開発大学校　　徳浜　元弘

An Implementation and Evaluation of the P2P Kademlia System Using the Tree Structure Type Distributed Hash Table

Motohiro TOKUHAMA

要約　P2P システムはネットワークのコンピュータ（ノード）同士が交信し、情報の共有やコラボレーション等を行うもので、その特徴は自己組織化が図れることである。本研究は P2P モデルのベースシステムとして木構造型の分散ハッシュテーブルを用いる Kademlia を取り上げ、その構成と実装に関する検討と評価を行う。また、定常時特性とルーティングアルゴリズムの性能に関するシミュレーション結果と、ノード検索の性能改善に関する試行も紹介する。

I　はじめに

コンピュータネットワークの接続形態として C/S（クライアント/サーバー）モデルと P2P（Peer to Peer）型モデルがある。

C/S 型モデルはサーバーに複数のクライアント（ユーザー）が接続し、サーバーの情報資源をクライアントが利用する形態で WWW や電子メールシステムなど広く利用されている。

一方 P2P 型モデルはネットワークのコンピュータ（ノード）同士が交信し、コンテンツ共有やコラボレーション等の機能を持つ分散コンピューティングネットワークシステムである。代表的なシステムに Napster[(1)]や Gnutella[(2)]がある。近年では P2P と言えば Winny による情報流出や、違法なファイル交換、不法コピーなどの温床になり、一般にさほど良いイメージは持たれていない。

一方、近未来での発展が期待されているユビキタスネットワークにおいては、その構成要素（ノード）が自立的に参加・離脱を行うために、管理面や処理性能の面で P2P システムが適している。

本論文はこのような P2P システムへの適用として木構造型の分散ハッシュテーブルを用いた Kademlia[(3)]を取り上げ、その構成と実装法について述べる。そして、シミュレーションを通してシステムの性能を評価する。また、ランダムなノード検索の性能改善のため、システムの総ノード数の推定に基づくアプローチを紹介する。

II　分散ハッシュテーブル(DHT)

分散ハッシュテーブル(Distributed Hash Table:以下 DHT と略す)は一種のインデックス情報でこれをシステムの構成ノードに分散して保持させてノードやコンテンツの検索に活用するものである。

DHT ではまず、各ノードに固有のノード ID を割り当てる。このノード ID は IP アドレスの文字列に SHA1 などのハッシュ関数を適用して算出する。

例えば、IP アドレスが “192.168.0.1” の場合、SHA1 で算出したハッシュ値　SHA1(“192.168.0.1”)= $(26ed8bbabb59013846d35b8c695ded7aa6efe712)_{16}$ がノード ID の数値として利用される。

また、コンテンツの格納先ノードもコンテンツのメタ情報（多くの場合はファイル名）に同一のハッシュ関数を適用してハッシュ値（キーと呼ぶ）を算出し、この値を格納するノードと関連付ける。関連付けの方法はいろいろな方式が提案されているが、より直感的な方法としては、得られたハッシュ値に近いノード ID を持つノードにコンテンツを格納することである。

利用者がコンテンツを取得するときはそのメタ情報のハッシュ値を求め、システム内の適当なノードにその居場所を問合せればよい（図１）。

分散ハッシュテーブルを使用した代表的なシステムに経路表形式を用いる Chord[(4)]がある。

論文受付日　　H25.4.26

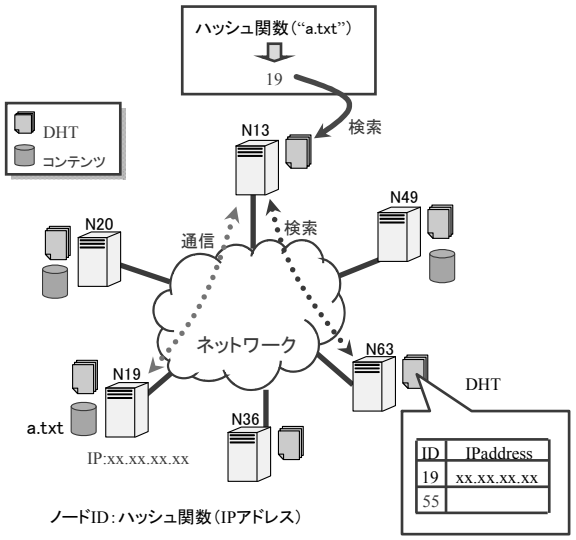

図1　DHT のイメージ

III　Kademlia

Kademlia[(3)]はMaymounkov らによって提案された木構造型の DHT を持つ P2P システムで BitTorrent[(5)]や eMule[(6)]等で実装されている。

1　Kademlia のノード空間

Kademlia のノード空間はノード ID が 160 ビット長で表現されるノードの集合から成る。ノード ID はノードの IP アドレスから求めたハッシュ値を用いる。コンテンツも、そのキーがハッシュ関数で 160 ビット長に変換され、キーに近い k 個のノードに格納される。

Kademlia の特徴の一つとして、ノード間の距離をノード ID の排他的論理和演算（XOR）を用いて決定することである。

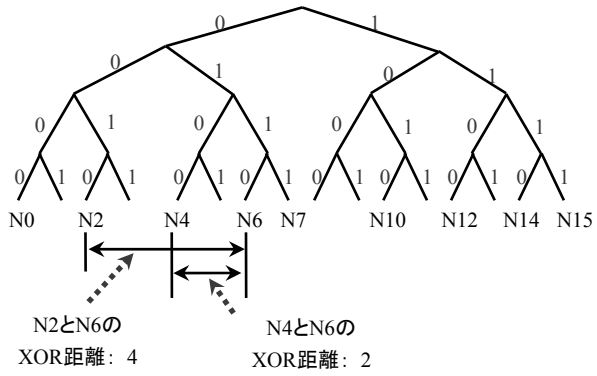

図2　Kademlia のノード空間

この XOR 距離によると、同じ値を持つ二つのノード ID の距離は 0 になる。また、二つのノード ID 値を 2 進数で表現したときに、先頭のマッチするビット列が長ければ長いほどそれらのノードは近接である。

図 2 は Kademlia のノード空間を図示した 2 分木で、説明の簡略化のため、ノード ID は 4 ビットとする。

2　Kademlia の DHT（k-バケット）

Kademlia の DHT はノード空間同様二分木構造であり、k-バケットと呼ばれる。また、k-バケットの葉要素であるバケットはノードオブジェクトを格納する。

その k-バケットの初期状態はひとつのバケットからなり、それがノード空間全体をカバーしている。そして他のノードからコンタクトがあるとそのノード ID を範囲として含むバケットに登録される。バケットの容量は無限ではなく、システム定数値 k が適用される。

バケットが満杯（k 個）になるとその範囲が自分自身のノード ID を含むときにそのバケットは二分割される。そうでなければそのコンタクトは破棄される。これによって Kademlia は DOS 攻撃への耐性を強めている。

図 3 はノード=$(0101\cdots)_2$ のバケットの分割例を示している。塗りつぶされたバケットはこれ以上分割されることはない。

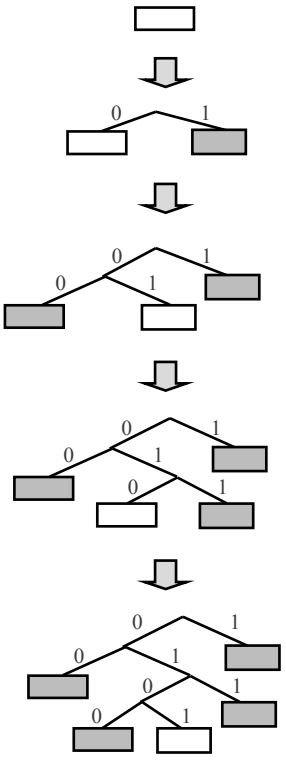

図3　バケットの分割例

こうして、k-バケットはノードがシステムに参加し、お互いのコンタクトを通して 1 個のバケットから成長していく。この場合のバケットの個数はシステム内のノード数を N としたときに高々$log_2 N + 1$ 個である。これはノードを二分木で表したときに、二分木の各レベルでバケットが作成され、自分自身のノードを含むバケットを加えて$log_2 N + 1$ となる。このバケット構造はあくまでも基本原理であり、次節ではさらに最適化される。

3 バケット分割の最適化

別の実装の最適化はすべてのバケットを一定のレベル b まで分割することである。k-バケットのサイズを$(2^b - 1)log_{2^b} N + 1$ にすることで、ノード検索時に発生する他のノードへのコンタクト数の期待値を$log_{2^b} N$に削減することができる。概念的には他のノードに保持すべき情報を自ノードに持たしてホップ数を削減することである。この最適化では、ノード ID を含まない範囲のバケットもb − 1レベルまで分割する。

例えば、b=2 ならノード ID を含まない ID 空間の半分は一回（二つの範囲へ）分割される。もし、b=3 なら、二つのレベルで最大四つの範囲を持つバケットへ分割される。

一般の分割規則は、バケットの範囲がノード自身の ID を含むとき、或いは二分木における k-バケットの深さ d が d≠0(mod b)なら、ノードは満杯のバケットを分割する。（深さ d はその k-バケットの範囲ですべてのノードで共有される先頭部分の長さである）

図 4 はノード ID が$(000000)_2$で始まるノードの k-バケットで、b=2 における分割の様子を図示したものである。二分木の深さ d が d≠0(mod 2)となるレベルのバケット数は高々3(=2^2-1)個まで分割され、末端のレベルは自分自身の範囲を含むので 4 個になる。

$(001)_2$の範囲を持つバケットは d=4 のレベルまで分割される可能性がある。塗りつぶされたバケットはこれ以上分割されない。

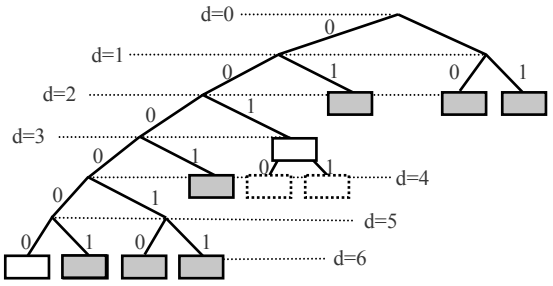

図4 バケットの分割例(b=2)

4 ノード検索

Kademlia のノード検索は目標 ID に近い k 個のノードを返す処理である。この処理はコンテンツを発行（アップロード）する際にキー値から格納するノードを決定するときにも使用される。

この検索処理は次のようなものになる。

①自身の k-バケットを探索して、検索対象 ID の範囲を含むバケットから k 個のノード ID を取り出す
②上記のノードに対して①と同様な処理の実行を依頼し k 個のノードを取得する
③検索対象 ID に等しいノード ID が返されるか、あるいはすべてのノード ID をチェックするまで②の処理を繰り返す

②は RPC(Remote Procedure Call)で実行される処理で、取得した k 個のノードに対してさらに目標 ID に近い k 個のノードを問い合わせる。

ノード検索の期待値は$O(log_{2^b} N)$ となる。これは図 4 でも示されるように検索の範囲を 1/2^bずつ狭めているからである。

5 ノードの参加と離脱

新しいノードがシステムに参加するには少なくとも一つのノード ID を知っておく必要がある。この既知のノードがシステムへ参加するためのブートストラップノードとなる。

ノードの参加は次のようにして行われる。

①参加するノードの ID に近いノードをブートストラップノードから k 個得る。
②それらの k 個のノードからバケットの内容をコピーし、プールする。
③プールしたすべてのノードをバケットに登録する。
④自分自身の存在を他のノードに知らせる

④の処理はバケットのリフレッシュと呼ばれるもので、各バケットの範囲に含まれるランダムな ID をバケット内のノードに対して検索させる。コンタクトを受けたノードは問合せ元を自分のバケットに登録する。これにより多数のノードにコンタクトさせてそれぞれのノードの k-バケットに自分自身を登録してもらう。バケットのリフレッシュは、さらに別の目的で使用される。それは、ノードの新規参加や離脱に対して k-バケットを最新の状態に保つためである。

ノードの離脱時には特にやるべきことはない。これもKademlia の大きな特徴である。他の形式のシステムの例を挙げると、経路表方式のDHTを採用しているChordシステムでは双方向リンクの修正、DHT の補正、コンテンツの移動等の処理が必要となり管理コストが増大する[8]。

6　コンテンツの発行・取得

Kademlia ではコンテンツのアップロードとダウンロードはコンテンツのメタ情報のハッシュ値（キー）とその内容（データ）に関連する操作である。

コンテンツのアップロード（発行）はまずキーに近いノードをk個取得し、そのk個のノードにコンタクトしてデータを転送する。複数のノードに同一データを格納することでノードの離脱やネットワークの障害等に対する冗長性を確保している。

また、コンテンツを取得するときも同様にキーに近いk 個のノードID を取得し、それらのノードに対してデータの転送を依頼する。データの取得はk個のノードすべてにコンタクトする必要はなく、一旦データが取得できればそこで完了する。

7　コンテンツの再発行

先に述べたように、コンテンツは冗長性を持たせるために、そのキーのハッシュ値に近い k 個のノードに格納される。これらはノードの参加、離脱に備えて定期的に再発行を行う必要がある。例えば、あるノードがシステムに参加し、そのノードID があるコンテンツのキーに近い場合、コンテンツはその新しく参加したノードにも格納されておかなければならない。

再発行はコンテンツの所有者側だけではなくコンテンツの発行を受けたk個のノードも実行する。

まず、コンテンツの所有者側のノードについては一定時間（例えば24h）おきにそれらを再発行する。所有者からの再発行を受けなかったノードに格納されたコンテンツは、陳腐化しているものとみなす。

また、コンテンツの発行を受けたノードも一定時間（例えば 1h）おきに格納されたコンテンツを再発行することになる。ここで懸念されるのは再発行の処理によってネットワークトラフィックが増大することである。単純計算でも一つのコンテンツあたり$(k-1)^2$ 個の再発行のメッセージが送出される。この問題を回避する方法はKademlia[3]に譲る。

Ⅳ Kademlia のシミュレーション

実際に Kademlia のシステムを作成して、定常時とノードの離脱・再参加時のノード検索の性能を計測してみた。使用言語はスクリプト言語のRuby[7]である。

1　シミュレーションの概要

(1)使用計算機のスペック

シミュレーションに使用したマシンのスペックは下記の通りである。

CPU	Intel® Xeon® E5-2670/2.60GHz
RAM	24GB
台数	1
OS	OpenSUSE 12.3 x86_64

(2)処理の概要

架空の IP アドレスを乱数で生成し、一定数（128～1024）のノードと一定数のバケットサイズ（k=4～32）を組み合わせてシステムを作成する。そして、これらのシステムに次のようなテストを行う。最適化のレベルはb=3 とした。

①総当たり方式によるノード検索
・システム内のノードから一つのノードxを取り出す
・同様に別のノードyを取り出す
・ノードyに、ノードxのノード検索を依頼する
・上記の処理をシステム内のすべてのノードに繰り返す
②ランダム方式によるノード検索
・ランダムなIPアドレスを持つノードを生成し、システム内のランダムに選んだノードに対して、ノード検索を依頼する
・上記の処理を一定数繰り返す
③ノード離脱後のノード検索
・一定数のノードをシステムから離脱させる
・残ったノード同士で①の総当たり検索を行う

2　システムプロパティ

(1)ノードの平均バケット数

それぞれのシステムの一つのノードで作成されるバケット数の平均値をバケットサイズ（k=4、8、12、16、32）毎に計測した（図５）。

バケットは二分木の 3 レベル毎に高々$7=(2^3-1)$ 個作成されるので、この値は$(2^b-1)\log_2{}^bN$ に近くなる。しかし、バケットサイズ k が大きいほど収容するノード数も増えるため、作成されるバケット数は少なくなる。

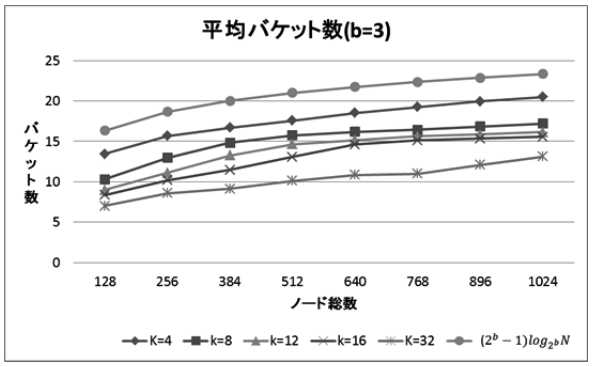

図5　ノードの平均バケット数

(2) k-バケットの収容ノード数

図 6 は k- バケットに収容しているノード数合計を計測したものである。その容量は k- バケットが収容できる最大値 $k(2^b-1)log_{2^b}N$ をかなり下回る。

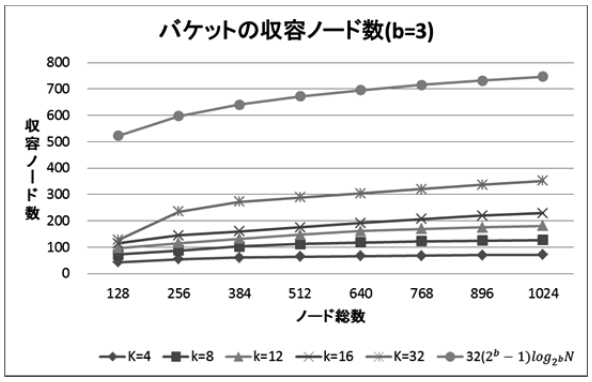

図6　バケットの収容ノード

(3) k-バケットの収容率

図 7 は k- バケットのノード収容率を表す。これは、(1)のバケット数を k 倍したノード数に対する収容ノード数の割合となる。k- バケットは満杯状態ではなく、10% 程度の空きがある。

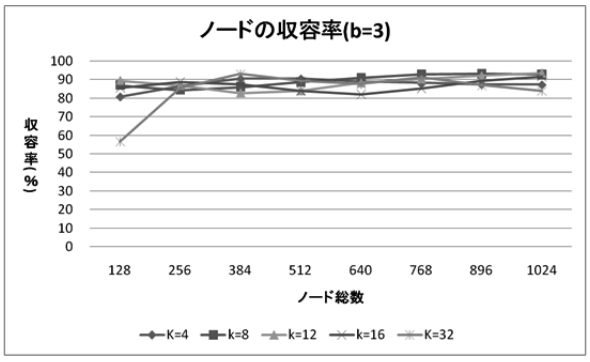

図7　バケットの収容率

(4) k-バケットの深さ

この数値は k-バケットの二分木の最大の深度（高さ）の平均を表したもので、バケットの接頭辞の最大値の平均となる。これらの値は log_2N を超えない。

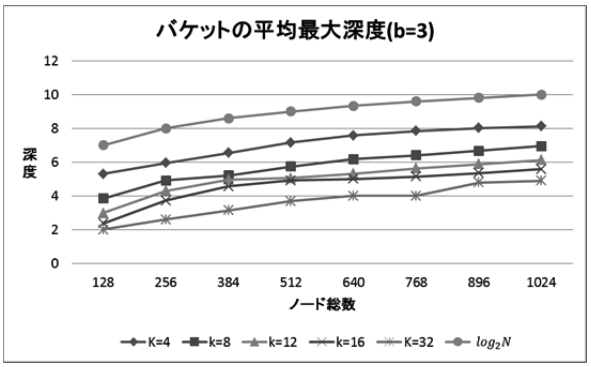

図8　バケットの平均最大深度

3　定常時でのノード検索

架空の IP アドレスを持つノードを一定数（128～1024）生成しシステムに参加させた後、総当り方式とランダム方式でのノード検索を実行し、コンタクトしたノード数を計測した。

(1)総当り方式によるノード検索

総当り方式ではシステム内のノード同士でルックアップを実行する。ノード数を N とすると N^2 回の検索が実行される。このテストは全て成功する。

総当りの結果は k の値が大きいほどノードコンタクト数は少なくなっている。これは k-バケットに多くのノードを収容できるため検索対象ノードを含む確率が大きくなるためである。

特に、k=32 の場合のノードコンタクト数は 1.0 未満である。これは、ノード検索が自ノードの k-バケット内だけか、或いは他のノードに対する 1 回のコンタクトで検索が完了していることになる。

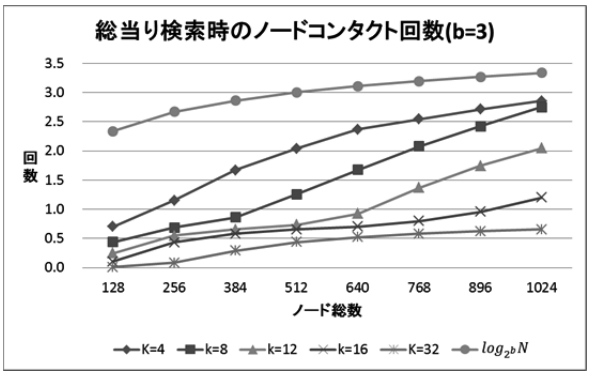

図9　ノードの総当り検索時のノードコンタクト回数

(2)ランダム方式によるノード検索

ランダム方式では総当りと同様に一定数のノードを

生成後に、故意に検索をフェイルさせるためにランダムなIPアドレスの検索を総当りと条件を合わせてN^2回実行する。

ランダムの場合は$log_{2^b}N$ の値をかなり上回っている。k の値が大きいほどその傾向は著しい。バケットに収容できるノードが増えるためより多くのノードにコンタクトを試みるからである。この結果は、P2P システムでのコンテンツ共有を考えると、無視することはできない。というのは、コンテンツの検索に同様のアルゴリズムを使用するために、システム内に存在しないコンテンツを検索しようとするとオーバーヘッドが大きくなることになる。

これを改善するには総当りによる検索が $log_{2^b}N$ を超えないことから、 c $log_{2^b}N$ を超えたら検索を停止するという方法が考えられる(c は定数)。

しかしながら、ノードの参加・離脱が頻繁に行われるという P2P システムの特性を考慮すると、リアルタイムにネットワーク上を横断してシステムのノード数 N を求めることは容易ではない。代わりに、ノードが保有している属性値から N を推定するアプローチが求められる。これについては、V 節で述べる。

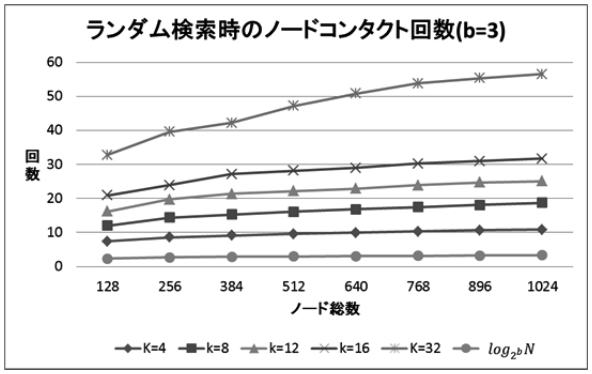

図10　ランダム検索時のノードコンタクト回数

4　ノード離脱後のノード検索

初期ノード総数として 1024 ノードとするネットワークを作成してから 128 ノードずつ 896 ノードまで強制的に離脱（タイムアウト）させ、生き残ったノード同士（128～128）で総当り検索を行いフェイルする割合を計測した。

ノードの離脱状態を表現するために、ノードの属性値として、keepalive 値を導入する。初期値を正数に設定しておき、ノードを強制的にタイムアウトさせるには、この値に非正数を設定する。従って、keepalive 値が非正数の場合はタイムアウトしているものと判定する。

図１１のグラフによると、k の値が大きいときには離脱の影響は少ない。特に k=32 の場合はフェイルしていない。総当たり検索の場合と同様に、k の値が大きくなると k-バケットに多くのノードを収容できるようになるため、フェイルしにくくなっている。

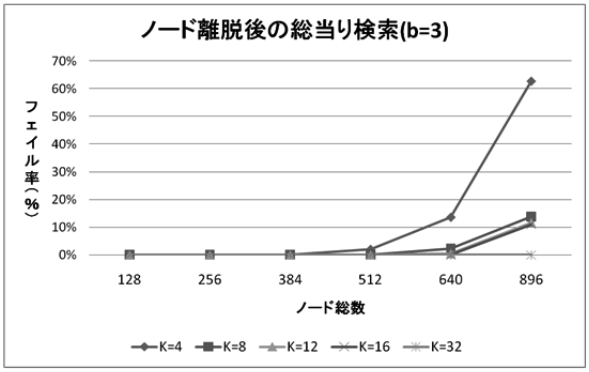

図11　ノード離脱後の総当り検索

V　システムの総ノード数の推定

ランダム方式によるノード検索の性能を改善する方法として、ノードの属性値からシステムのノード数 N_k^b を推定し、これをノード検索のリミッターに活用することを試みる。

図 5 のグラフより、ノードの平均バケット数は $(2^b-1)log_{2^b}N$ より小さいことに着目して、次の式(1)を設定する。B_k^b はノードのバケット数とする。

$$B_k^b < (2^b-1)log_{2^b}\frac{N_k^b}{k}+1 \quad \cdots\cdots(1)$$

k-バケットの木構造において、一つのバケットに k 個のノードが収容されるため、 $log_{2^b}N$ 個の節ではなく、 $log_{2^b}\frac{N_k^b}{k}$ 個の節において高々(2^b-1) 個のバケットが作成されることを考慮して(1)式を設定した。

(1)式をN_k^b について解くと次式が得られる。

$$N_k^b > k(2^b)^t \quad \cdots\cdots(2)$$

但し、$t=\frac{B_k^b-1}{2^b-1}$ とする。

再度シミュレーションを行い、各ノードのバケット数を(2)式に適用して得られた、総ノード数の推定値 N_k^b の平均値を図１２に示す。

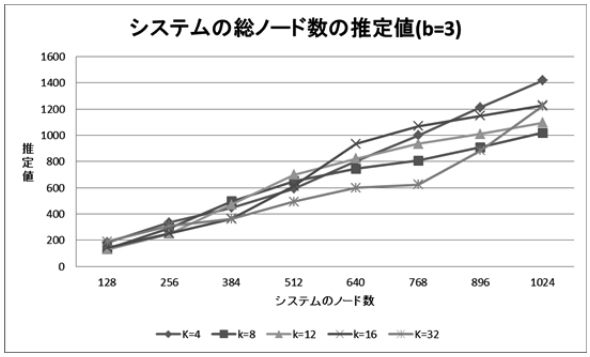

図12　システムの総ノード数の推定値

図１２によると、k の値が大きくなるほど、システムのノード数との誤差のばらつきが大きい。

次に、$log_{2^b}N_k^b$ の値を算出してグラフで表したものが図１３である。このグラフからすると、対数のカーブは$log_{2^b}N$ とほぼ重なっており、図１２の誤差のばらつきの影響は小さくなっている。

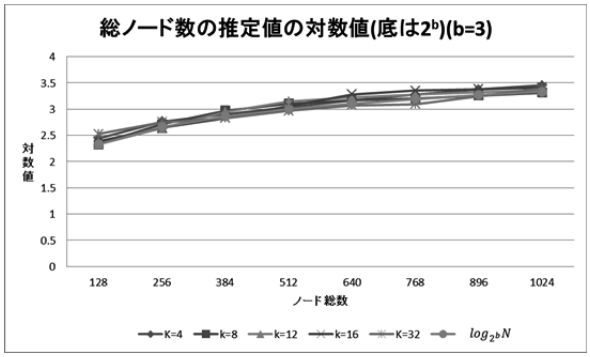

図13　総ノード数の推定値の対数値（底は 2^b）

これらの結果から、c $log_{2^b}N_k^b$ の値をランダムなノード検索のリミッターとして採用することにする（c は定数）。これを使って実際にシミュレーションした結果が、図１４のグラフである。

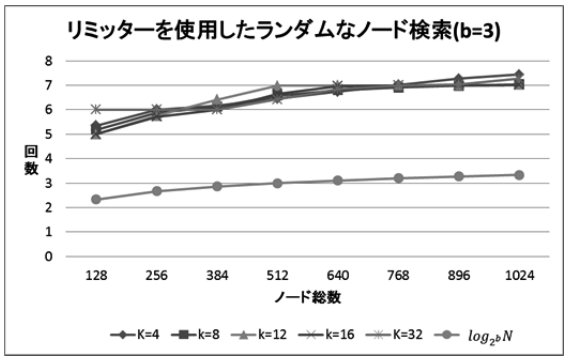

図14　リミッターを使用したランダムなノード検索

このシミュレーションでは、c の値は 2.0 とし、リミッターは c $log_{2^b}N_k^b$ を整数値に切り上げた数値にした。各ノード検索において、ノードコンタクト回数がこのリミッター値に達したら、ノード検索を中止する処理を組み込んだ。

図１４のランダム検索の結果は、ノードコンタクト数が 10 未満になり、図１０の k=32 の場合と比べるとノードコンタクト数はおよそ 10%～20%に削減された。

VI　おわりに

本論文では、動的にネットワークへノードの離脱と再参加が頻繁に繰り返されるユビキタス環境を想定し、二分木構造の DHT を持つ Kademlia が P2P 型自己組織化ネットワークシステムとしての適用性が高く、スケーラビリティがあることを示した。

特に定常時の総当りノード検索でのノードコンタクト数は $O(log_{2^b}N)$ で高速であり、ノード離脱時での総当り検索でもフェイルしにくいという特性を持つ。さらに、バケットサイズ(k)が大きいほどノード検索の性能はよい。

しかし、故意にフェイルさせるランダム検索では逆にノード検索効率が悪いという結果も明らかにし、改善策も示した。

今後の課題としては次のようなものが挙げられる。

・実用レベルのノード数を想定したシミュレーション

・バケットリフレッシュとコンテンツの再発行時におけるトラフィック軽減化の検討

[参考文献]

(1) ネットワーク管理者のための Napster 入門、http://www.atmarkit.co.jp/fwin2k/experiments/napster_for_admin/napster_for_admin_1.html

(2) ネットワーク管理者のための Gnutella 入門、http://www.atmarkit.co.jp/fwin2k/experiments/gnutella_for_admin/gnutella_for_admin_0.html

(3) Maymounkov. P., and Mazieres. D., Kademlia: A peer-to-peer information system based on the XOR metric. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems, Springer-Verlag version, Cambridge, MA, Mar. 2002

(4) Ion Stoica, et.al, Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications IEEE/ACM Transactions on Networking, Vol, 11, No, 1, Feb 2003, pp.17-32

(5) BitTorrent, http://www.bittorrent.com/

(6) eMule, http://www.emule-project.net/
(7) オブジェクト指向スクリプト言語 Ruby、
http://www.ruby-lang.org/ja/
(8) 徳浜元弘,中沢実,服部進実,　ユビキタスネットワークにおける分散ハッシュテーブルの構築と評価,
情処研報　Vol.2006, No26, 2006, p215-220.