DHTシステムの仕組みと性能と課題について

報告者 ： 駒井秀一

2008 年 9 月 30 日

株式会社ライブドア ネットワーク事業部
通信環境技術研究室

1　はじめに

当報告書では、インターネットシステム、オーバレイネットワークの概念を整理し、その基礎技術として分散ハッシュテーブルと呼ばれる手法を紹介する。分散ハッシュテーブルを用いたオーバレイネットワークの具体的な一例として、数値的に探索空間の分割を行う Chord[1]と、経路表を用いる Pastly[2]について詳細に解説する。また、分散ハッシュテーブルの応用例として、Pastry アルゴリズムをベースにしている BambooDHT[3]実装方法や、実装時の動作について解説する。

分散ハッシュテーブルに注目が集まる要因の１つは簡潔さや効率性の高さによる。
分散システム在り方の利点である、耐障害性、規模拡張性、可塑性を想定する上で、冗長性の確保が必要であることは容易に想像できるところだが、分散ハッシュテーブルでは、ハッシュテーブルという考え方をネットワーク上に拡張し、原則的に代替することを可能とする各機能をネットワークに参加するノード間で分担する事により、負荷の分散を行える点で冗長性を共有できる。

また、IP システムアーキテクチャにおけるグローバル空間内での到達可能性は、ユーザが自由に活動する目的を達成する必要条件にあたる。分散ハッシュテーブルのアーキテクチャデザインにて、人気のあるコンテンツを保有している特定ノードへの大きなアクセス負荷を見直し、より公平な分散ハッシュテーブルネットワークを構築する可能性を検討する。以上を当報告書の目的とする。

2　現代インターネットアーキテクチャ

近年、FTTH(Fiber to The Home)や DSL(Digital Subscriber Line)をはじめとするブロードバンドの急進展や、複製/記録/保存などのハード面での劇的なコスト低下により、情報流通の性質には変化が生まれてきた。

単位ユーザあたりのトラフィック量の増大は、都市部などのエリアを限定でするものではなく、P2P などを利用する極端なユーザや、企業ユーザに限定した状況とはいえなくなってきた。平行して、インターネットに流通するコアコンテンツとは、音や動画などの重量マルチメディアコンテンツへシフトが加速した。現代流ともいえる、この情報流通の側面を垣間見れば、その中には IP システムという１つの共通したアーキテクチャデザインに、以下であげられるような排他性を否定する自由な情報の流通側面を共通して確認できる。

- ハード/ソフト両面でオープンなシステム。
- 代替的で、選択肢が多様であること。
- 競争原理がありコスト性や協調性に富む。

これら性質を一貫して保持し、事業や経営の変化にも強く、利用者や利用方法を単調に制限しない自由な情報流通が可能な特徴を持つことになったのであろう。つまり、IP システムは自律的なシステムの提供が実現出来ているわけだ。
インターネットがグローバル規模なアーキテクチャとして成長維持される背景であるこれら要因を踏まえ、IP システム上へ展開されるオーバーレイネットワークという存在が技術と政策の両面で持続的提供が可能なシステムアーキテクチャの１つである旨の認識へと到達する。また、オーバーレイネットワークは分散性という特徴も併せ持つ。インターネットの原型とされる ARPANET が分散型システムであったように、オーバーレイネットワークにおいては一部に破壊や障害があったとしても全体としての最低限の機能は維持される機構を持つ。また、ネットワークの規模レベルは、原則として中央コンピュータの性能に比例する側面を持つため、ネットワークにおけるノードの数や通信量に応じて増加

するステートの管理の負荷にも悩まされる事がない。コンピュータの技術向上は日進月歩であり、コスト面でも性能面でも輝かしい成長をとげてはいるが、ノード数や通信量に応じた中央コンピュータの拡張に追いつくものでもないだろう。中央集権タイプのアーキテクチャデザインでは、システム規模は一定の範囲内で収束するだろう。一方、分散型システムにおいては、それぞれのノードマシンが参加する通信の量というものは中央コンピュータ程のものではない。分散的側面をもつオーバーレイネットワークが注目を集めるに値するアーキテクチャであることは妥当なわけだ。

3　DHT オーバレイネットワーク

前章で既に述べたインターネットシステム拡張の背景を踏まえると、透明性の高い基盤の上に自由なプロトコルやトポロジを定義、形成可能なネットワークを構成することはとても大切な考え方の1つであろうという結論に到達するだろう。

ここでは、コンテンツを流通するにあたり、中央集権タイプのアーキテクチャデザインであるクライアントサーバアーキテクチャと分散性を持つ P2P アーキテクチャという両者の大きな差異点について考える。

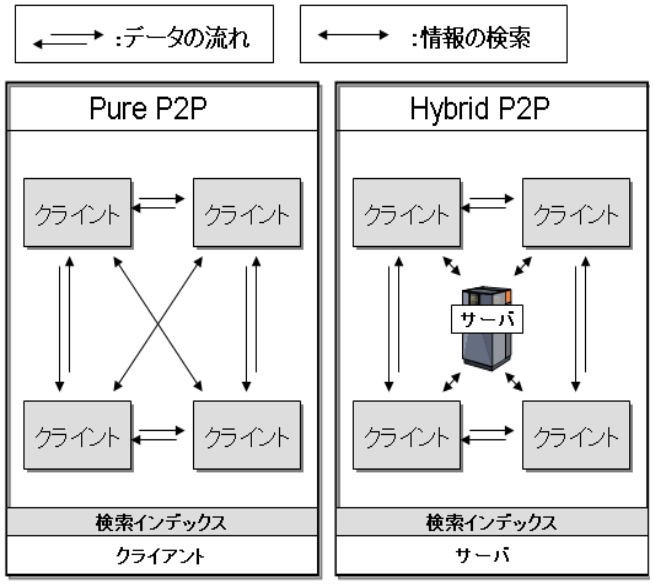

図 3.1　Hybrid P2PとPure P2P

両者の最大の違いはサーバやクライアントは１つの点である必要が無く、ISP も企業もユーザも、ネットワークに参加する全ての機器がサーバであり、クライアントにもなりうる事である。それ故、サービスプロバイダが主役となるクライアントサーバモデルとは異なり、耐障害性、拡張性、可塑性を実現する対等な peer 同士の協調作業を実現する。

P2P アーキテクチャについては、その検索方法の違いからクライアント・サーバモデルを融合させサーバを使用する「ハイブリッド P2P」 と、完全な分散環境である「ピュア P2P」 に分類できる。透明性の高い基盤の上に自由な「エンド・ツー・エンド」 なサービスを展開可能にする P2P アーキテクチャは、自律性や分散性を持ったスケールフリーなアルゴリズムであると言える。今回、当報告書ではオーバレイネットワークの１つのシステムデザインとして、分散ハッシュテーブル(DHT)に着目し報告を行う。

3-1　クライアント・サーバモデルの問題点

先に説明を行った通り、現状のネットワークで主流となるクライアント・サーバモデルは単一サーバであるため，クライアント数の増加に伴いサーバの負荷が増大し、中継ネットワークでボトルネックが生じてしまう。また、サーバに障害が起きた場合、サービス全体が停止してしまう。

そこで解決策に Peer-to-Peer(P2P)が挙げられる。P2P ではピアと呼ばれる端末が、サーバとクライアントの両方の機能を持ちホスト上でパケットの複製・配信が行うことができる。これにより、サーバの負荷分散、帯域の有効利用が可能になる。

3-2　peer to peer(P2P)モデル

ピアという単語には互いに等しいという意味が潜在するので、対等なレイヤでそれぞれの協調的な活動実体が考えられる。ピア同士の協調作業を想定する上では、動的に変化する幅の広い条件に依存せずに分散システムが全体として現状の性質を維持できることが必要である。当条件の実現を視

野に、システムが冗長的であり、互いにオープンでリカーシブルに相互通信を実現するノードで構成がされ、状態や機能面でのキャッシュ機能が出来る必要性がある。

■　耐障害性の実現

サーバに障害が発生した場合、その直後から何ら通常と変化の無いサーバコンテンツの送受信を行うためには、該当サーバのキャシュが存在し、互いがトランスピアレントな状態であることが必要条件といえる。P2P モデルでは、耐障害性を実現できる。

■　再帰性の実現(リカーシブル)

分散システム内の一部が消失してしまった場合、オーバーレイネットワーク内の残数のノードネットワークが、消失前の状態を保持するために、消失した代替的システムを複製実現する事が必要条件といえる。また、再帰性が確実に実現されるために、システム内に指揮全般を取り仕切るスーパーノードが存在するのではなく、システム内のそれぞれが自律的に再構築に参加し、回復への道が進行する必要性がある。P2P モデルでは、耐障害性を実現できる。

3-3　分散ハッシュテーブル

表 3.3　分散ハッシュテーブルの特徴比較

	Chord	Pastry
Path Length	Log_2N	log_bN
NeightborState	Log_2N	log_bN
Insert	$O(Log_22N)$	$O(log_bN)$

先に述べたオーバーレイネットワークの基礎技術として、分散ハッシュテーブル(DHT)が提案されている。その具体例として、Chord、Pastrly が例に挙げられるが、それらアルゴリズムの特徴を次章及び表 3.3 にて示し解説する。分散ハッシュテーブルを用いたネットワークにおいては、key 値に対応するデータ(value)の取得を行う事が可能であり、これを Lookup と呼ぶ。この Lookup 処理では、一般的にいうフレーズ検索のような「あいまい検索」や「前方一致」といった検索処理ではなく、key に対応するデータ(value)が一意に決定される。これが通常取り扱われている「検索」機能と「Lookup」との違いに該当する。 Lookup で取得されるデータ(value)は、検索結果であるコンテンツ自体の場合も想定できるが、IP アドレスのリストなどが記録されており、コンテンツの場所を指し示すためのポインタとしても用いられる。実装次第では検索主が行うコマンド処理は 1 回にすることも可能だが、プロセスとしては、コンテンツの IP アドレスのリストを取得し、その後に当該 IP アドレスへ直接に接続し、コンテンツを取得するという 2 回の処理を行うステップが要される。

4　分散ハッシュテーブル

4-1　Chord アルゴリズム

分散システムにより検索を実現する方法として Chord ネットワークを解説する。Chord アルゴリズムは、"Ion Stoica" 氏[4]らにより提案されたオーバレイネットワークの 1 つである。

4-1-1.　ID 空間

Chord では、ネットワーク全体が ID サークルという円状の仮想空間として定義され、全てのノードと全てのデータは、この ID サークル上に配置される。

Chord のネットワーク空間の大きさは、最大 2^m 個のノードから構成される。Chord では、ノード及びデータを ID サークル円状均等に配置するにあたり、ハッシュ関数(SHA-1)を用い、160 bit のうち m bit (1 ≦m ≦160; m ∈ N)を ID 空間に使用する。

・ノード ID：SHA-1(IP アドレス)
・key：SHA-1(キーワード)

160bit の ID 空間の場合、$3.40*10^{38}$ の ID が存在する事になり、コンテンツと IP アドレスをマッピングしても衝突し

なことが前提とされる。もし衝突する場合には、任意の数字とノード ID を加えた値(anyID + SHA-1(ip))が用いられる。Chord の ID 空間がリング状につながっており、ノード ID と keyID がランダムに分散した状態となっている。
ここでは、説明しやすいように 7bit(0-127)の ID 空間を用いて説明を行う。距離の定義は ID の単純な差である。ネットワークの近さを全く考慮しない下層的なネットワークである。従って、Chord のトポロジーは数直線の最小値と最大値を結んだ円形である。各ノードはノード ID に基づき ID サークル上に配置される。

4-1-2 参照処理

Chord の参照処理はどのように行われるのかについて述べる。Chord での探索は ID が大きくなる方向に順に処理が実行される。あるノードの存在を考えた場合、ノード ID が大きくなる方向で見た場合に最近接のノードを successor といい、それとは反対に、ID 値が小さくなる方向で最近接の ID を predecessor という。また、Chord では以下に解説する 3 種類のルーティングテーブルを用いる。これらのルーティング情報を持つことで、最終的に各ノードは ID サークル全体をカバーすることになり、任意の問い合わせに対して、適切なルーティング処理を行う事ができる。

まず 1 つは successor リストと呼ばれるルーティングテーブルである。successor リストは後続ノードのルートを r (r ∈ N) 個だけ保持する.
各 interval に含まれるオブジェクト ID のオブジェクトを検索する際に、次にどのノード(successor ノード)に検索要求をフォワードすべてきかが示されている。ただし、intareval とは具体的に次式で定義される時計回りの区間をさす。

interval = [start, end)
start =$(N + 2^{k-1}) \bmod 2^m$
end =$(N + 2^k) \bmod 2^m$
※ N = 当該ノード ID

2 つ目のルーティングテーブルは precedessor のルートを持つルーティングテーブルである。
3 つ目は以下に解説する finger テーブルと呼ばれるルーティングテーブルである。
finger テーブルは、
自分より 2^k (0 ≦ k ≦ m -1; k ∈ N) 個離れたノードを保持するテーブルである。これを使用することにより、1 ホップ毎に探索範囲を少なくとも 2 つに分割しながらノードを絞っていくので、ノード数全体を N とすると探索のホップ数は O(logN)相当 になる。

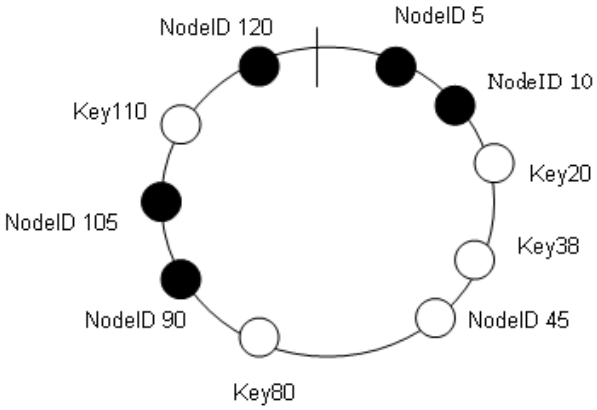

図 4.1 keyIDのマッピング

参照時の処理の流れを図を使用し解説する。図 4.1 では、ノード 5→ノード 10→ノード 45→ノード 90→ノード 105→ノード 120→ノード 5 の様に接続される。この接続はルータ同士の接続ではなくノード同士で作られる接続である。
そのため、ノード間には複数のノードが存在し、ノードに格納され、このノードを代表ノードと呼ぶ。key20、key38 は時計回りに最も近い node45 に格納される。同じように、key80 のデータは node90、key110 は node120 に格納される。そして、key の値とデータのペアは、ネットワークに参

加しているノードのみに保存される。

図 4.2 は基本的な参照方法を表している。探索者である node45 が、key110 に対応するデータを取得した場合、node45→node90→node105→node120 へと検索クエリが転送される。

node120 は keye110 を保持しえるため、key110 に対応するデータを node45 へ送信する。

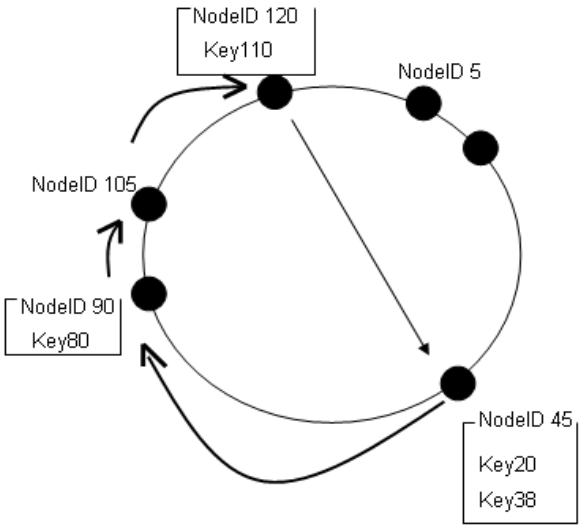

図 4.2　基本的な参照処理(key80 を探索)

4-1-3　join(追加)

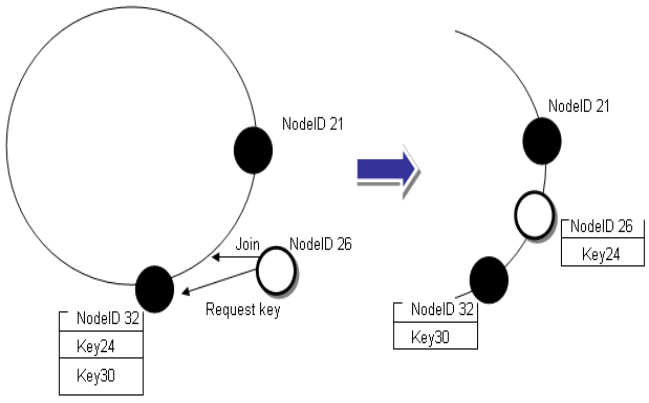

図 4.3　ノード参加時の様子

Chord ネットワークにおいてノードの追加及び削除があった場合の処理について説明する。このとき最低 1 ノード、既にネットワークに参加しているノードと通信できる状態であるとする。

参加ノードは自ノードのハッシュ key を計算し、既存ノードと通信を行う事で、自分が Chord の円上のどこに位置すべきか判断を行う。次に、 successor と predecessor のルーティングテーブルを利用して、自分のルーティングテーブルを構築する。合わせて successor から自分が担当するべき HashTable の要素を受け取る。最後に、自分が参加した通知を Chord のネットワークに転送させることで、他ノードのルーティングテーブルを更新する。

図 4.3 は ID が 26 のノードが参加する場合を示している。ノード 21 とノード 32 の間に位置することが分かるので、ノード 26 がしなければいけないことは優先順に次のようになる。

(1)ノード 26 の successor list にノード 32 を追加。
(2)ノード 32 から、ノード 26 が保有すべき key 集合を譲り受ける。
(3)ノード 21 の successorlist にノード 26 を追加。
(4)ノード 26 の finger table を構築。
(5)ノード 26 を finger table に持つべきノードへ finger table の更新依頼。(理論上，最大 160 個)

ノード 32 の探索は、ハッシュキー26 に対するキー保有ノードの探索そのものであり、successor list の更新は，近傍ノードの間の通信で済むのでコストは低い。finger table の構築は理論的には 160 回の探索を行う必要がある。しかし、ノード 32 の finger table を参照することで、探索の多くを省略できる。他ノードの finger table を更新することは更に面倒で、2 の i 乗ずつ半時計まわりに遠いノードへ更新依頼を行う必要がある。更新依頼を受けたノードは，必要なら finger table を更新する。更新依頼は半時計まわりに伝播を続ける必要がある。

4-1-4　Chord の特徴

ノード数 N の場合を想定した Chord ネットワークにおける諸性質を以下に示す。

・ネットワークの維持には他の O(logN)ノードの情報を所持する必要がある。
・O(logN)のメッセージ数で参照が可能である。
・join/leave する際のメッセージ数は $O(\log^2(N))$

Chord では参照やネットワークへの join/leave にサーバを必要とせず、各ノードが完全に分離して処理が進行する。検索可能ノード数 N の増加に伴い線形のパケット増加が起こる。一方、Chord ではパケット数の増加は O(logN)、ホップ数は O(logN)にすぎず、規模拡張性に優れている。また、Freenet[5]などにおいては、検索時にデータが存在しない事を判断するには、検索時の遅延を推測しデータの不在を判断していたが、Chord ではデータが存在しない場合であっても、明示的にデータが存在しないことをしることが可能である。

前述の通り 1 ノードを参照するには、O(logN)メッセージ数(ホップ数)がかかる。そのため、ネットワークの参加へは自ノードを参照される O(logN)個のノードに変更を行うため、参加時に生じるメッセージ数は $0(\log^2 N)$となる。

4-1-5　Chord API

・insert（key, value）:ネットワークに key と value を挿入する。
・lookup(key) : key に対応する value を返す。
・update(key,value) : key と newval をネットワークに挿入する。
・join() : Chord ネットワークに自分を追加する。
・leave() : Chord ネットワークから退出する。

4-1-6　考察

Chord ネットワークは、ハッシュ関数を用いて各ノードに規則的に役割を持たせている。そのため、アドホック的なネットワークに比べ状態遷移が多く、ネットワークの維持にかかる処理は複雑化することが予想される。

しかし一方、分散アプリケーションでは、どこにデータを格納するか問題となるが、その解決方法の 1 つとして、分散ハッシュテーブルである Chord を用いた参照処理が利用できる。そのため、今後様々な分散アプリケーションに適用できると考えられる。

5　Pastry

BambooDHT が利用するアルゴリズムである Pastry の前に、Pastry アルゴリズムの土台となっている Plaxton[6]構造について説明する。
Plaxton は経路表を用いる同様に経路表を用いるオーバレイネットワークの関連研究に、マイクロソフト社/ライス大学による Pastry である。
その後、Pastry アルゴリズムの一例として有名な BambooDHT に関して、情報の少ないその具体的な実装方法を解説し、サンプルコードの解説や検索挙動についての説明も合わせて行う。

5-1　Plaxton 構造

Plaxton 構造では、全てのノードがサーバであり、ルータであり、クライアントである。つまり、オブジェクトを格納し、経路制御を行い、探索要求を発行できる。他の分散ハッシュテーブルアルゴリズムと同様に、Plaxton 構造でも全てのノード及びオブジェクトに識別子が割り当てられており、SHA-1 のハッシュアルゴリズムにより、均等に ID 空間に分布する機構となっている。

Plaxton 構造における、オブジェクト毎にそのネットワーク上の位置を知る。探索の際には当該ノードを頂点とする

tree 構造を逆に辿るというものである。この際の経路制御は、各ノードと近隣ノードとの論理的なリンクを表す経路制御を参照する事により行われる。経路表のエントリには、与えられた目的地の識別子とプレフィックスが最長一致する近隣ノードに中継する事で最終的に目的地に到達する。

この方法では、N 個の可能な要素から成る名前空間で、識別子お基数を b として解釈する場合、最大でも logN ホップで目的のノードに到達でき、経路表のサイズは b・logN の固定サイズとなる。

遅延を短縮し、帯域の有効活用を促進すると共にシステム信頼性を向上する上で最も重要である。
(通信の経路が長ければデータが失われたり、壊れる可能性が増大するし、データやサービスが近傍に存在していれば、世界のほかの部分から分断されたとしてもアクセスできる)

5-2-1　Pastry アルゴリズム

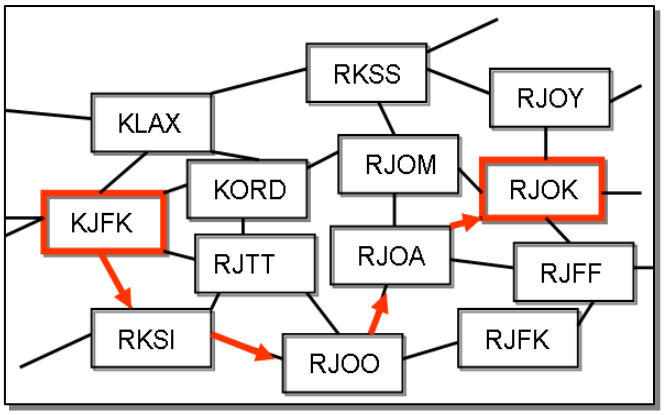

図 5.2　Plaxtonにおけるルーティング例

Pastry はマイクロソフト社とライス大学で開発された DHT の実装を目的とした P2P システムである。目的の ID を探索するために、Plaxton のアルゴリズムを採用している。メッセージを目的の ID へルーティングするためには、目的の ID とプレフィックス長が最長一致した近隣のノードへ選択し、そのノードを中継するといった処理を行う。
例えば、ノード KJFK を出発点として、ノード RJOK に到達するためには以下の手順をたどる。具体的には、それぞれのルーティングステップでメッセージを、プレフィックスが最長一致する ID を選び出す処理は、上位から 1 行ずつ文字を揃えていく処理に相当する。

■　ID が R***であるノード
　→　ID が RJ**であるノード
　　→　ID が RJO*であるノード
　　　→　ID が RJOK であるノード

これ以上のプレフィックス長の一致する key がない場合は、メッセージは現在のノードと同じ長さの key プレフィックスを共有するが、数として自身の持つ key より近いノード ID を持つノードに転送を進める。

Pastry ではノードに 128bit のハッシュ値から構成されるノード ID が割り当てられる。そのノード ID は、どこか特定の中央サーバから割り当てられる ID ではなく、各ノードが独立に計算して取得する ID による。Join(参加) する際にノードの IP アドレスなどから暗号ハッシュをランダムに割り当てる。また、N 個のノードが存在する場合、logN ステップ以下で与えられた key にメッセージを送信する事ができる。Plaxton アルゴリズムに従うルーティングテーブルとはまた別に、「leaf set」と「neighbor set」となる補助的なテーブルを保持している。leaf set については、当該ノードの近辺に存在するノードの ID を保持するための表である。距離が近いノードへの経路を、通常の経路表とは別に一定数保持し探索を効率化する役目を持つ。探索においては、まず始めに leaf set を参照するため、leaf set を適切に維持する過程はトポロジの適切な維持に重要な役目を果たす。テーブル内で保持される対象ノード ID の数については実装時に設定する事が可能である。

neighbor set には、ネットワーク内での物理的に近い距離にある対象ノードの ID を保持してる。ハッシュ値の論理空間上の距離とは別の尺度として、IP として近傍のノードの経路を保持している。leaf set 同様に実装時に設定を変更することが可能である。Plaxton と比較した際、IP としての物

理的距離の親和性をルーティングの際に取り入れることが出来る点が、大きく異なる点である。以上より、例えば以下のようなルーティングを繰り返して目的のノードへ到達することになる。

(A) 対象とするkeyが、leaf set内のノードIDが保持する範囲に含まれているかを確認し、距離として最近接と考えられるノードに対してメッセージを転送する。
(B) leaf set内に対象ノードの存在を確認できなかった際、保持するルーティングテーブルより転送を行うノードを探索し、対象ノードへメッセージの転送を開始する。
(C) ルーティングテーブルにさえ、対象のノードを確認できなかった際、リーフセットの中から改めて自分よりは対象ノードにIDとして距離の近接するノードを探索し、当該ノードに対してメッセージの転送を開始する。
(D) IDとして近接するIDが1つも存在しない際、自ら目的ノードとなりうる。

前述の通り、Pastry でルーティングに要するホップ数は(1/b)logN、ルーティング・テーブルのエントリー数は$[(2^b-1)/d]$logNと考えられる。(但し、Nは全ノード数とする)

5-2-2 Join

新規ノードが join する際、テーブルを更新し他のノードへの通知をする必要がある。新規ノードは物理的近傍にあると仮定するノードAを記憶していた場合、自らノードIDを自律的に決定しマルチキャストを用いて参加位置を判断する。新規ノードのノードIDを仮にXとした場合、Xと等しいkeyと共にjoinメッセージを転送しノードAに対して依頼を行う。ノードXと数字的に最も等しいノードをノードZとすると、joinメッセージを受け取ったノードAとZ、及びAとZの間にあるノードはノードX にテーブルを送信する。新規ノード X は送信されたテーブルを基にテーブルを次の方法で初期化する。つまり、ノードXの経路表構築であると同時に、ノードAからノードZの経路表の更新処理も兼ねているのが確認できる。

・新規ノードX は neighbor set に関してノードA のものを使用する。

・ZはXに近いノードID を持っているので、leaf set はノードZ ものを使用する。

・次に、ルーティングテーブルを考慮する。最後にノードXは作成し終えたテーブルを neighbor set・leaf set・routing table の作成に用いたノードに対して複製を送信する。複製を受信したノードはその情報を元に自分のテーブルに更新を行う。

反対に、離脱ノードが発生した場合はどうなのか。各のノードは定期的に生存メッセージをテーブル内のノードに送信している。そして leaf set のノードの離脱の場合には leaf set 内のノードに、neighbor set のノードの場合は neighbor set のノードに問い合わせを行い埋め合わせをする。また routing table の場合には、例えば1行目にあるA列目のノードが離脱した場合、1行目の他のノードに問い合わせる。その問い合わせを行ったノードの 1 行目 A 列目を埋め合わせに用いるノードにする。1 行目 A 列目が空の場合は(L+1)行目のノードに問い合わせ、そのノードの1 行目 A 要素のノードで、離脱分のノードの埋め合わせを行う。

Routeing table

-0-*******	1	-2-*******	-3-*******
0	1-1-******	1-2-******	1-3-******
10-0-*****	10-1-32102	2	10-3-*****
102-0-****	102-1-****	102-2-****	3
1023-0-***	1023-1-***	1023-2-***	3
10233-0-**	1	10233-2-**	
0		102331-2-*	
1023310-0-		2	

表 5.3　テーブル情報例

5-3-1　BambooDHT

Pastry アルゴリズムをベースにしたフリーな実装としては Bamboo と FreePastry が有名である。どちらも Java で実装がされている。　BambooDHT は Pastry アルゴリズムをベースにしながら、そこに留まらず独自拡張も続けている DHT システムだ。Pastry の派生と見なして考えることができるだろう。FreePastry は Pastry アルゴリズムの実装を提供することが目的である。そのため Pastry アルゴリズムの評価や研究が目的である場合には FreePastry が適している。一方、DHT の開発参加や新機能の実装などが目的である場合は、BambooDHT を利用するのが適切であろう。

5-3-2　BambooDHT の実装

BambooDHT の実装方法をここに解説する。当報告書の執筆時点で、正式にリリース番号がついているのはバージョン 1.0.1 である。しかし、v1.0.1 のソースコードはとても古いため、今後、CVS の最新ツリーまたはスナップショットを使うべきであると考える。　本報告書では、2006 年 3 月 3 日の CVS スナップショット版を使っている。2004 年末以降、Bamboo のソースコードは Java1.5 ベースになっている。Bamboo の最新版をコンパイルするには、JDK1.5 以降の要件を満たす必要がある。Bamboo のコンパイルについては、次のように環境変数 JAVAHOME の設定対応を行い make 処理を実行するのみで完了される。

(1)BambooDHT の公式サイトよりソースをインストールしコンパイル作業を行う。

```
[*console*] BambooDHT のコンパイル方法
$ export JAVAHOME=/usr/local/1.6.0_07
$ tar zxf bamboo-cvs-2006-03-03.tgz
$ cd bamboo
$ make                                    [*console*]
```

また、bamboo ディレクトリ下にある Makefile 内には、bamboo の home ディレクトリを規定する箇所がある。念のため bamboo ホームディレクトリが、bamboo をインストールしているディレクトリに設定されているかの確認を済ませておく必要がある。

(2)動作のテスト

BambooDHT には正常なインストールや動作を確認するための簡単なテストモジュールが下記ディレクトリに用意されている。表示されるメニューに従い複数のノードを立ち上げ、ノードステータスなどを確認する。

```
[*console*] BambooDHT の動作テスト
$ cd /bamboo/test
$ perl -w location-test-menu.pl
Menu:
  1. Check node status
  2. Check object pointers
  3. Check for exceptions
  4. Start a node

Your choice: [4]　4
 Startinglocalhost:3630withgatewaylocalhost:3630.
cfg=/tmp/experiment-22277-localhost-3630.Cfg
log=/tmp/experiment-22277-localhost-3630.log
pid=22282

Your choice: [4]　1
/tmp/experiment-22277-localhost-3630.log:Tapestry:
 ready
/tmp/experiment-22277-localhost-3633.log:Tapestry:
 ready
[*console*]
```

(3)BambooDHT の基礎動作を確認する。

BambooDHT を動作させるためにあたっては、BambooDHT

ネットワークを構成するノードひとつずつに設定ファイルを用意する必要がある。当コンフィギレーションファイルを通じ、ノードの BambooDHT への動作要件を定める。例えば、エントリーポイントに該当するブートストラップノードの指定や、参加時の自身の IP アドレスやポート番号の指定を行う。最もシンプルな設定ファイル例を次に示す。

[*code*] シンプルな設定コンフィギレーションファイル例

```
<sandstorm>
<global>
<initargs>
node_id localhost:3630 # このノードのエントリポイントです。この場合 UDP の 3630 ポートを待つことになる。
</initargs>
</global>
<stages>
<Network>
class bamboo.network.Network
</Network>
<Router>
class bamboo.router.Router
<initargs>
gateway localhost:3630 # このノードが DHT に参加するためのブートストラップノードを指定する。
digit_values 16
</initargs>
</Router>
</stages>
</sandstorm>
```

[*code*]

ブートストラップノードに自分自身を指定した場合1台のノードだけのネットワークになる。2 つ目以降の別ノードがこのノードを指定すると、ネットワークが広がっていく。上記の設定ファイルを/tmp/node1.cfg として保存する。BambooDHT を以下のコマンドで実行できる。

[*console*] BambooDHT の実行例

```
$ bin/run-java bamboo.lss.DustDevil /tmp/node1 .cfg
Sandstorm: Ready
INFO bamboo.router.Router: Joined through gateway 127.0.0.1:3630
Tapestry: ready
```

[*console*]

ノードをもうひとつ起動して、このノードとネットワークを構築することも可能になる。

2 つ目のノードを立ち上げるならば、前述の通り 2 つ目のノードの動作を定義するための別のコンフィギレーションファイルを作成する必要がある。例えば、ローカルホストのポート 3631 で 2 番目のノードを実行させるならば、設定コンフィギレーションファイルは以下の通りになる。

[*code*] 2 番目のノードのコンフィグ例

```
<sandstorm>
<global>
<initargs>
node_id localhost:3631
</initargs>
</global>
<stages>
<Network>
class bamboo.network.Network
</Network>
<Router>
class bamboo.router.Router
<initargs>
gateway localhost:3630 #
digit_values 16
</initargs>
</Router>
</stages>
</sandstorm>
```

[*code*]

このコンフィギレーションファイルを「/tmp/node2.cfg」というファイル名で保存したとすると、以下のコマンドで 2 番目のノードを動作実現させることが出来る。

[*console*] BambooDHT で 2 つ目のノード起動例
$ bin/run-java bamboo.lss.DustDevil /tmp/node2.cfg
Sandstorm: Ready
779 INFO bamboo.router.Router: Trying to join through gateway 127.0.0.1:3630
131 INFO bamboo.router.Router: Joined through gateway (127.0.0.1:3630, 0x19d5e887)
Tapestry: ready
163 INFO bamboo.router.Router: added 127.0.0.1:3630 to leaf set
169 INFO bamboo.router.Router: added 127.0.0.1:3630 to routing table
[*console*]

2 番目のノードの経路表と leaf set に、1 番目のノード(localhost:3630)が追加されたことが確認できる。
一方、1 番目のノードのコンソール出力を見ると、その経路表と leaf set に 2 番目のノード(localhost:3631)が追加されたデバッグメッセージを見つけることができる。ノードは 2 つしかないが、DHT のネットワークを構築できたことになる。この要領で設定ファイルを作成してノードを増やしていけば、理論上はいくらでも巨大な DHT ネットワークを構築が可能である。

5-3-3　BambooDHT プログラミング

BambooDHT プログラミングには主に 2 種類が想定できる。1 つは、BambooDHT のクラス API を使ったプログラミングであり、DHT の機能を組み込んだアプリケーションを作ることができる。もう 1 つは、BambooDHT ネットワークに対して、通信プロトコルでアクセスするプログラミングである。Web に例えると、BambooDHT のクラス API を使うプログラミィングは Web アプリケーションサーバ上で行うプログラミングであり、もう片方の後者は Web サービスにおいて API を呼び出すプログラミングであると考えられるだろう。当報告書では、BambooDHT のクラス API を使用したプログラミング方法を例に説明を行う。DHT の key をベースにした value の登録(put)と取り出し(get)を自動的に行う事を実現する。コード例は別紙(巻末)のようになり、「/Bamboo_home/src/test」ディレクトリに「Pgbd.java」として保存した場合の例になる。
「Pgbd.java」では次のような key と value を DHT ネットワークに登録する。この登録の可否は、以下のコマンドで確認できる。

・Put する Key：ld_pics
・Get する Key：ld_wireless
・PutするValue：http://pics.livedoor.com/002/3JPG
・TTL:3600

[*console*] 登録項目の確認

$ bin/run-java bamboo.dht.Get localhost 3632 ld_pics
・
・
http://pics.livedoor.com/002/3JPG が表示される。

■ ファイルを１つエントリした後のノード状態

・Node1 の状態

[node info]
Hostname:localhost
IP：127.0.0.1
port：3630
ID：0x19d5e887
Current Strage:91 Byte

[Leaf Set]

ip	port	ID
127.0.0.1	3633	0x76663e70

[Routing Table]

ip	port	ID
127.0.0.1	3633	0x76663e70

・Node2 の状態

[node info]
Hostname:localhost
IP：127.0.0.1
port：3631
ID：0x76663e70
Current Strage:91 Byte

[Leaf Set]

ip	port	ID
127.0.0.1	3633	0x19d5e887

[Routing Table]

ip	port	ID
127.0.0.1	3633	0x19d5e887

6　今後の課題

以下に述べるような問題を認識しており、以後我々は、それら問題点を解決する手法を研究し、実験による評価および考察を行い、情報の爆発的な増加やコンテンツの多様化などに伴い、スケーラブルかつコンテンツの頻繁な更新にも耐えうる P2P ネットワークの構築を目指す。

一般にデータへの検索要求の発生頻度は Zipf[7]の法則に従うことが知られている。一部のデータにアクセスが集中するというデータ要求の分布を示したという経験則で、アクセスには大きな偏りがあることが実環境では起こっていることが WEB ページのアクセスランキングや、動画投稿サイトの再生数など様々な事柄に当てはまる法則である。当現象については、自社 blog サービスなどでも同様の現象を確認できる。

これらアクセス頻度の偏りによる一極的で不公平な負荷を想定し、要求データのアクセス頻度に応じて、複製配置確率を変化させ、確率的に複製配置を実施していくなど、アクセスの集中負荷を分散させ、トポロジー全体としての安定性と効率性を向上させるためのる DHT 発展的拡張課題を持つ。

$$q_j = \frac{j^{-\alpha}}{\sum_{m=1}^{K} m^{-\alpha}}$$

k は総データ数。α はアクセス確率を決定するための Zipf 係数であり、この値が大きい程一部のデータが頻繁に要求される事が分かっている。j はランクを示す。

またコンテンツの更新においては、古いコンテンツを残しておいたまま新しいコンテンツを追加しているが、従来の DHT の研究ではコンテンツの取得・公開・削除の動作にばかり着目し、更新の動作への配慮が欠けている。例えば、更新前のコンテンツと更新後のコンテンツを同じ key に割り当てた場合、取得者はコンテンツを取得するまでそれらを判別できない。一方、異なる key に割り当てた場合には、元の key では更新後のコンテンツを取得できないため、取得者はなんらかの手段で新しい key の知識を手に入れなくてはならない。

そのため、従来の DHT はコンテンツを頻繁に更新するサービスには適していない。従って，今後 DHT を実用化していくうえでこの問題が妨げとなることが考えられる。また、key や value の更新が発生する際には、都度の新しい key や value の情報に関するオーバーレイネットワーク上での更新対応が必要となる DHT の特徴より、更新頻度の高い blog など CGM サービスに適用する場合を仮定すると、その負荷は極端に大きなものであると想定できるため、コンテンツ情報の更新取得時におけるトラフィクの最小限にするための効率性を向上させるための拡張課題を持つ。

7 まとめ

各データに対するアクセス分布やネットワークトポロジなどの環境情報や、ピアグループを意識したオーバーレイネットワークを構成することにより、システムの耐故障性や規模拡張性、可塑性を改善できる余地がある。

デザイン上、サーバへの問い合わせが集中する箇所の存在を見直し、目的とするデータの検索性能を損なうことなく、ストレージに発生する負荷をネットワーク全体に分散する事を可能にする手法を検討する事を当報告書の次期検討課題とし報告を終える。

[謝辞]
本報告を行うにあたり、日ごろより日頃から議論させていただいている通信環境技術室の皆様に感謝する。

[参考文献]

[1] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan (2001). “Chord: A scalable peer-to-peer lookup service for internet applications”. ACM SIGCOMM Computer Communication Review 31 (4): 149 - 160. DOI: 10.1145/964723.383071

[2] Pastry:Pastry : Scalable,decentralized lbject location and routing for large-scale peer-to-peer systems (http://research.microsoft.com/~antr/PAST/pastry.pdf)

[3] Bamboo : http://bamboo-dht.org/

[4] Ion Stoica : http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[5] Freenet : http://freenetproject.org/

[6] PLAXTON, C., RAJARAM, R., AND RICHA, A. W. Accessing nearby copies of replicated objects in a distributed environment. In Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), June 1997.

[7]G. K. Zipf:“Human Behavior and the Principle of Least Effort”, Addison-Wesley(1949).

[5-3-3　BambooDHT プログラミング例]

```
package test;

import java.io.*;
import ostore.util.*;
import bamboo.dht.*;
import bamboo.util.StandardStage;
import bamboo.lss.ASyncCore;
import bamboo.router.Router;
import seda.sandStorm.api.ConfigDataIF;
import java.security.MessageDigest;
import java.util.Random;
import java.math.BigInteger;

public class pgbd extends StandardStage {
    protected MessageDigest digest;
    protected GatewayClient client;
    public void init(ConfigDataIF config) throws Exception {
        super.init(config);
        try {
            digest = MessageDigest.getInstance("SHA");
        } catch (Exception e) {
            assert false;
        }
        String client_stg_name = config_get_string(config, "client_stage_name");
        client = (GatewayClient)lookup_stage(config, client_stg_name);

        acore.register_timer(1000, do_put_cb, null);
    }
    public ASyncCore.TimerCB do_put_cb = new ASyncCore.TimerCB() {
            public void timer_cb(Object user_data) {
                String key = "ld_pics";
                String val = "http://pics.livedoor.com/002/3JPG";
                bamboo_put_args put_args = new bamboo_put_args();
                put_args.application = "my-test";
                put_args.key = new bamboo_key();
```

```
            put_args.key.value = digest.digest(key.getBytes());
            put_args.value = new bamboo_value();
            put_args.value.value = val.getBytes();
            put_args.ttl_sec = 3600;// [sec]

            client.put(put_args, curry(put_done_cb, put_args, null));
        }
    };
public Thunk3<bamboo_put_args, Object, Integer> put_done_cb =
    new Thunk3<bamboo_put_args, Object, Integer>() {
        public void run(bamboo_put_args put_args, Object user_ctx, Integer put_res) {
            if (put_res.intValue() == bamboo_stat.BAMBOO_OK) {
                System.out.println("----------------------------------------");
                System.out.println("put success!");
                System.out.println("----------------------------------------");
                acore.register_timer(1000, do_get_cb, null);
            } else {
                System.out.println("DHT put failure. reason=" + put_res);
            }
        }
    };
Public ASyncCore.TimerCB do_get_cb = new ASyncCore.TimerCB() {
        public void timer_cb(Object user_data) {
            String key = "ld_wireless";
            bamboo_get_args get_args = new bamboo_get_args();
            get_args.application = "my-test";
            get_args.key = new bamboo_key();
            get_args.key.value = digest.digest(key.getBytes());
            get_args.maxvals = 1;
            get_args.placemark = new bamboo_placemark();
            get_args.placemark.value = new byte[] {};
            client.get(get_args, curry(get_done_cb, null));
        }
    };
public Thunk2<Object, bamboo_get_res> get_done_cb =
    new Thunk2<Object, bamboo_get_res>() {
```

```
            public void run(Object user_ctx, bamboo_get_res get_res) {
                if (get_res.values.length > 0) {
                    System.out.println("DHT get value is " + new String(get_res.values[0].value));
                } else {
                    System.out.println("DHT not found");
                }
            }
        };
}
```