

構造化オーバーレイでの一括フォワーディング

首 藤 一 幸[†]

構造化オーバーレイにおいて、IP 網などアンダーレイの負荷を軽減し、同時にメッセージ配送を効率化する手法、一括フォワーディングを提案する。オーバーレイに対して多数のメッセージ配送を要求する場合、例えば DHT に対して多数の put, get を行う場合に、配送経路上のノードが複数の要求をまとめて扱うことでオーバーレイでのフォワーディング回数を減らす。これによって、オーバーレイでのスループットとノードの処理負担が改善され、アンダーレイでの通信回数も減る。1,000 ノードからなる DHT に対して get 要求を 10 ずつ束ねて行うことで、通信回数は 34% ~ 12% まで減った。所要時間は逐次に配送した場合の 13.0% ~ 9.7% となった。

Collective Forwarding on Structured Overlays

KAZUYUKI SHUDO[†]

This paper presents *collective forwarding*, a technique to alleviate an underlay network such as an IP network and improve efficiency of message delivery on structured overlays. The technique improves communication throughput, reduces loads of nodes on an overlay, and reduces the number of communications on an underlay by bundling a number of delivery requests on an overlay. It shows its performance in case that an overlay delivers a large number of messages, for example we put or get many items on a DHT. It reduced the number of transmissions to 34% ~ 12% and the time to get items to 13.0% ~ 9.7% by bundling each 10 get requests on a DHT with 1,000 nodes.

1. はじめに

構造化オーバーレイ (structured overlay) は、中心となるサーバ的なノードのない、全ノードが対等な非集中かつ自律的な分散環境において、ID に基づいたメッセージ配送を行う仕掛けである。メッセージ配送機構 (key-based routing) の上に、分散ハッシュ表 (DHT) やマルチキャスト、エニーキャストといった機能を構成できる¹⁾。スケーラビリティや耐故障性の高さから基盤的な分散システムの基礎技術として期待され、たとえば DNS への適用検討²⁾ やコンテンツ配信への実際の応用^{3),4)} が始まっている。

N ノードからなる構造化オーバーレイでは一般に、メッセージ配送に $O(\log N)$ 回のフォワーディングを要する。その各フォワーディングが、アンダーレイ、たとえば IP 網でのパケット配送であり、そのパケットは複数のルータを経由して目的地に到達する。このように、オーバーレイ上のメッセージ配送は、アンダーレイでの一対一直接通信よりも重い処理となり、所要時間

もアンダーレイへの負担も大きいものとなる。

ここで、複数の宛先 ID に向けてメッセージを配送する状況を考える。もし、それぞれの経路に重なりがあれば、次ホップへフォワードするメッセージを 1 つにまとめることで、総フォワーディング回数を削減できる。これにより、オーバーレイでのスループットが改善され、フォワーディングを行うノードの処理が軽減される。また、アンダーレイでの通信回数が減り、ルータなどの処理が軽減される。加えて、経路が重なるか否か、つまり、次ホップが同一か否かを判断する必要から、複数のメッセージ配送要求を一括して扱うため、総所要時間も短縮される。

DHT の応用では、多数のメッセージ配送要求がいちどきに生じ、前述の状況が起きる。DHT 応用、例えば DNS²⁾ にせよ RFID タグにせよ、格納するデータの数は数百万やそれ以上のオーダーとなる。初期稼働時やバックアップ時には、その全体もしくはある部分を一度に put/get する必要が生じる。その際、DHT を構成する構造化オーバーレイでは多数のメッセージ配送要求がいちどきに生じる。

経路の重なりは、偶発の発生を期待するのではなく、積極的に起こす。つまり、重なりそうな宛先 ID の組

[†] 東京工業大学
Tokyo Institute of Technology

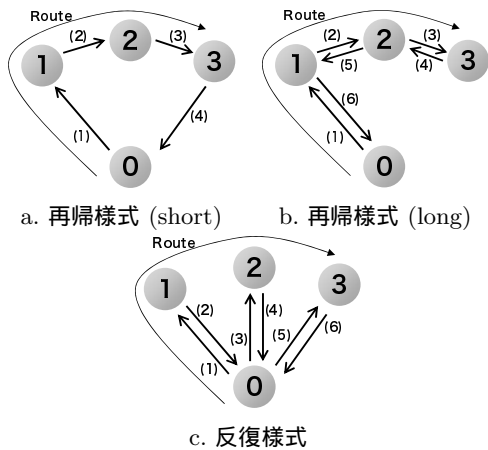


図1 フォワーディング様式
Fig.1 Forwarding styles.

を探して、それらをひとまとめにし、一括してオーバーレイに対するメッセージ配送要求を行う。

本論文で我々は、こうして、次ホップが同一となる複数のメッセージを一括してフォワードする手法、一括フォワーディング (*collective forwarding*) を提案する。提案手法は、オーバーレイ構築ツールキット Overlay Weaver^{5),6)} に実装し、その上で評価した。4章で、通信回数と配送所要時間についての評価結果を示す。5章では関連研究を示し、6章で今後の展望を述べる。

2. 一括フォワーディング

提案手法、一括フォワーディングを説明する。

構造化オーバーレイのメッセージ配送では、宛先は ID で表現され、その宛先 ID を担当するノード (responsible node) にメッセージが配送される。経路上の各ノードは、経路表に基づいて宛先 ID に応じた次ホップを決定し、次ホップにメッセージをフォワードする。具体的には、IP 網といったアンダーレイに対して次ホップへの配送を要求する。フォワードと言っても、反復様式 (iterative forwarding/routing/lookup) (図1)^{7),8)} の場合、次ホップに対して直接通信を行うのは配送要求元ノード、つまり図1のノード0だが、本論文では、図1cのノード1から2へ、2から3へのリダイレクト、これも含めてフォワーディングと呼ぶ。

ここで、複数のメッセージ配送要求をまとめて取り扱うことを考える。それぞれの宛先 ID は異なっていてよい。あるノード上で各宛先 ID について次ホップを決定した結果、次ホップが同一ノードとなるメッセージの組があれば、それらはひとまとめにして次ホップにフォワードできる。これによってフォワーディングの回数を減らせる。メッセージがまとめられ送受信の

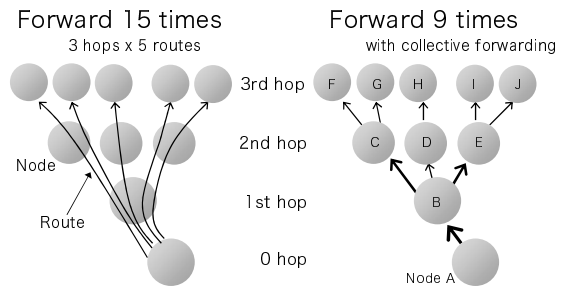
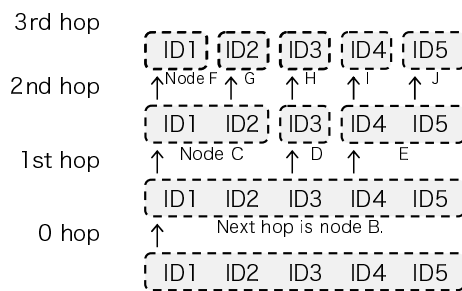


図2 フォワーディング回数の削減
Fig.2 Reduction in the number of times forwarding performed.



A bundle is partitioned on each hop according to next hops of target IDs in it.

図3 次ホップに応じた bundle の分割
Fig.3 Bundle partitioning according to next hops.

単位が大きくなることで、スループットの向上が見込める。同時に、フォワーディングごとに必要となるメッセージ処理の回数が減るため、それに由来するノードの処理が軽減される。また、アンダーレイでの通信回数、ひいては負担が減る。加えて、複数の配送要求をまとめて、並行して扱うため、配送の総所要時間も短縮される。こうして、複数のメッセージをまとめて一度に次ホップへフォワードする手法を、本論文では一括フォワーディングと呼ぶ。

図2に一括フォワーディングの具体例を示す。左は宛先 ID ごとに個別に配送を行った場合、右が一括フォワーディングを適用した場合である。ノード間の結線は1回のフォワーディングを表す。5つの宛先 ID に向けて配送を行い、それぞれ経路長が3となっている。この場合、通常配送 (左図) では $3 \times 5 = 15$ 回のフォワーディングが必要であるところ、一括フォワーディングによって9回にまで削減される。

以下に一括フォワーディングの手順を示す。本論文では、複数のメッセージ、もしくは宛先 ID 群をまとめたものを bundle と呼ぶ。

- (1) 配送要求元ノードは、全宛先 ID を単一の bundle に含める。

- (2) bundle 内の全宛先 ID について次ホップを決定する。
- (3) 次ホップに応じて bundle を分割する。つまり、次ホップが同一ノードである宛先 ID 群を 1 つの bundle にまとめる。
- (4) bundle ごとに、次ホップにフォワードする。
- (5) (2) に戻る。

図 3 は、図 2 の状況で、各ホップにおいて bundle がどう分割されるかを示している。配送要求元、つまりノード A では、まず、全宛先 ID を単一の bundle に収める。続いてノード A は、全宛先 ID について次ホップを決定し、その結果すべてノード B となるため、分割せず、単一 bundle のまままとめてノード B にフォワードする。ノード B では次ホップがノード C, D, E と分かれるため、bundle は 3 つに分割され、各ノードにフォワードされる。以下同様である。

この提案手法によって、オーバーレイでのスループットの改善、ノードの処理軽減が期待できる。一般に、小さな単位で通信したのでは高いスループットを達成することは難しい。例えば IP 網では、ルータが TCP ACK の 40 バイトという小さなパケットでワイヤスピードを達成できる場合であっても、上位の TCP や UDP では、小さな単位で通信したのでは高いスループットは達成できないことが知られている。提案手法は、メッセージをまとめて大きくすることでスループットを改善し得る。フォワーディングの際、各ノードは受け取ったメッセージを解釈し、組み立てて送出する。この処理の回数はフォワーディングの回数に比例する。処理の負担と時間はオーバーレイのプロトコル次第であり、定量的な議論には多くの前提が必要だが、一般に、注目に値する大きさとなる。提案手法はフォワーディング回数を減らすことで、ノードの処理を軽減する。

アンダーレイに対しては、提案手法は通信回数を減らし、ひいては負荷を軽減する。例えば IP 網では、ルータがパケットごとにヘッダを解析して転送先ポートを決める。パケット転送のボトルネックはパケット 1 つごとに行われるこうした処理であり、このことは、ルータの処理能力が pps (packets per second) で表されることに表れている。提案手法は通信回数、つまりパケット転送の回数を減らすことで、ルータの処理を軽減する。

提案手法によって減るのは通信回数であって、トラフィックではない点に注意されたい。複数のメッセージをひとまとめにするという手法であるため、各メッセージのアンダーレイ上での経路と通信量は変わらない。メッセージヘッダ中の宛先 ID 以外の部分は bun-

dle 化によって総量が減るが、宛先 ID とボディのサイズに比べれば影響は微々たるものだろう。

ただし、提案手法で (オーバーレイ) マルチキャストを行う場合、状況は異なる。すなわち、配送要求元が同一のメッセージボディに対して複数の宛先 ID を指定し、提案手法が、そのメッセージを全宛先 ID に対して配送する場合である。その場合、複数の宛先 ID に対してメッセージボディは 1 つとなるので、全宛先 ID に対して個別にユニキャストを行う場合よりもボディサイズ $\times (N - 1)$ (N は受信ノード数) まで通信量を削減できる。一般的なマルチキャスト手法では、受信メンバの管理、配送木の構築などによって、送信前に事前に受信者を確定しておく。それに対して提案手法を用いた small group マルチキャストでは (基盤となる構造化オーバーレイさえ構築されていれば) そういった事前準備が不要であり、送信のたびに宛先 ID 群を指定できるという特徴がある。

3. 宛先 ID の bundle 化

bundle は、いったん一括フォワーディングが始まれば、分割されて小さくなる一方である (図 3)。一方で、配送すべきメッセージ群、つまり宛先 ID 群を与えられた状況で、最初にそれらをどう bundle 化するか、という問題は残されている。ここでは、bundle のサイズと、最初の bundle 化を行うタイミングを論じる。

bundle はあまり大きくするべきではない。なぜなら、アンダーレイによっては、送受信できるパケットのサイズに制限があったり、また、そうでなくとも大きいことで信頼性の問題が生じたりするからである。例えば IP の場合、MTU (最大パケット長) より大きいデータグラムは分割され、分割数が多いほどデータグラムが失われる可能性が高くなる。

bundle のサイズを抑えるためには、bundle に含める宛先 ID の数を制限する必要がある。反復様式の場合、配送途中ではメッセージボディこそ送受信されないが、それでも、bundle 内の宛先 ID 群はやりとりされる。構造化オーバーレイでは ID 長は 128 ビット以上であることが多く、例えば 100 の ID を bundle に含めると、それだけで 1,600 バイト以上となってしまふ。最適な bundle サイズ (含む ID の数) は、ID 長やメッセージボディの長さに依存し、これをどう定めるかは今後の課題である。

では、bundle サイズの上限決め、つまり最初の bundle 化はいつ行うべきか? アンダーレイ上での送受信サイズを抑えるためには、配送要求元による最初の

フォワーディングまでには行う必要がある。この、最も遅いタイミングで制限する場合、配送要求元が第1ホップを決定した後で bundle 化を行うことができ、bundle 数、ひいてはフォワーディング回数を最小にできる。

しかし今回は、オーバーレイに対してメッセージ群を渡す時点ですでに bundle 化しておく、という方針を採った。これは、DHT で言うと、数千数万のキーを一度に put/get するのではなく、一定数、例えば 10 や 20 ずつ put/get する、ということにあたる。この設計は、bundle 化の処理はオーバーレイのノードに行わせるには高度過ぎる、という著者の判断に基づく。

一括フォワーディングの過程での bundle 分割は、次ホップに応じたグループ化であり、工夫の余地もないごく軽い処理である。しかし、サイズ制限を目的とした最初の bundle 化は、そのやり方によって一括フォワーディングの効果が大きく変わってくる。経路の重複が少なく bundle サイズがすぐに 1 になってしまったのでは、通信回数削減効果は得られない。効果を得るべく、重複がなるべく起こるように bundle 群を構成する処理はすなわちクラスタリングである。ということは、処理が重いというだけでなく、さまざまな手法の間にトレードオフがあり、やり方が自明でないということである。

この種の高度な機能を基盤システムにどうサポートさせるかは魅力的な研究課題である。しかし、知られた経験則に従うなら、基盤部分に含めることは避けるべきである。一方で、オーバーレイ上のノードだからこそ可能な最適化、例えば、ノード群の ID をある程度把握した上で、それに応じたクラスタリングも考えられる。これは残された課題である。

4. 評価

メッセージ配送に要する通信回数と所要時間について提案手法を評価した。本章でその結果を示す。

提案手法は、Overlay Weaver^{5),6)} (以下、OW) に実装し、その上で評価した。OW は構造化オーバーレイ構築ツールキットであり、ID に基づくメッセージ配送機構と、その上の DHT およびアプリケーション層マルチキャスト機能を提供する。

OW の場合、提案手法は routing driver 部に実装することになる。他のコンポーネント、例えばルーティングアルゴリズムに手を入れる必要はない。そのため、OW が提供するアルゴリズム、Chord、Kademlia、Koorde、Pastry、Tapestry と反復/再帰フォワーディングのすべての組み合わせで、提案手法を利用で

きる。

実験は、OW が提供する分散環境エミュレータの上で複数の DHT シェルを動作させて行った。このエミュレータは、計算機 (Java 仮想マシン) 1 台上で多数のノードを動作させ、それらノードを与えられたシナリオに従って制御できる。ここで動作するコードは、実機上、Internet 上でそのまま動作する。つまり、本章の実験で用いた実装は実地でもそのまま動作する。

実験には、2.8 GHz Pentium D プロセッサ、x86-64 用 Linux 2.6.25、x86 用 Java 2 SE 5.0 Update 15 の HostSpot Server VM を用いた。OW の版は 0.8.7 である。すべての実験は 5 回行い、最良値と最悪値を捨て、残り 3 回分の値を平均したものを結果として採用した。

4.1 宛先 ID のクラスタリング

提案手法からより高い効果を得るためには、含む宛先 ID 群の経路がなるべく重なるように bundle を構成する必要がある (3 章)。

提案手法の可能性を調べるために、クラスタリング処理自体の効率よりも、なるべく良い結果を得ることを優先した。そのため、クラスタリングに要する時間は度外視し、次に示す単純な手法を採用した。

- (1) 空の bundle を用意する。また、最後に直前の bundle に加えた ID を目標 ID とする。1 つ目の bundle に対しては、0x00...0 を目標 ID とする。
- (2) 未処理 ID 群から、目標 ID に対して最も距離が近い ID を選び、bundle に移す。
- (3) 未処理 ID 群から、bundle 内の全 ID に対する距離の平均が最も小さい ID を選び、bundle に移す。
- (4) (3) を、bundle のサイズ (ID の数) が上限値に達するまで繰り返す。
- (5) サイズが上限値に達した bundle は、そこで確定する。
- (6) (1) に戻る。

ここでの距離とは、ルーティングアルゴリズムによって算出方法が異なる、ID 間の数値的な距離を指す。例えば、ID 3 から 4 への距離は、Chord であれば 1 であり、XOR 距離を使う Kademlia であれば $3 \oplus 4 = 7$ となる。つまり、提案手法およびその OW への実装はルーティングアルゴリズムを選ばないが、適切な初期 bundle 構成はアルゴリズムごとに異なる。

構造化オーバーレイにおける ID 間距離には方向があり、そのため、知られた多くのクラスタリング手法はそのままでは適用できない点に注意されたい。例えば

Chord において、ID 3 から 4 への距離は 1 であるが、4 から 3 への距離は $2^{160} - 1$ (ID が 160 ビットの場合) となる。

このクラスタリング手順は、クラスタ内の全 ID に対する距離が最短、つまりその時点では最良に見える ID をクラスタに追加していくという単純なものであり、良い結果が期待できる。

その代わり計算量のオーダは高い。ID の総数を N として、それと比較して bundle のサイズが十分に小さいと仮定すると、およそ $O(N^2)$ となる。2.2 GHz の Athlon 64 3500+ を用いて、10,000 の宛先 ID を Chord, Koorde 向けにクラスタリングする処理に 91.1 秒、Kademlia 向けに 106.8 秒、Pastry 向けに 116.7 秒、Tapestry 向けに 2,444.3 秒を要した。もっとも、DHT に対して put/get すべきキーが時間を追ってやってくる状況では put/get のためのフォワーディングと、続いて put/get すべきキーのクラスタリングは並行して行うことができ、クラスタリングに要する時間を隠蔽できる。また、宛先 ID を一定数ごとに分割してクラスタリングすることで、所要時間の短縮を図れる。例えば、10,000 の宛先 ID を 1,000 ずつに分割して 10 回処理することで、Chord, Koorde 向けには 14.4 秒、Tapestry 向けでも 238.2 秒まで短縮できる。加えて、よく知られたクラスタリング手法、例えば k-means 法や階層的な手法を元にした手法を用いることで、よりオーダの低い計算量でクラスタリングできる可能性がある。

4.2 通信回数

DHT に対する put および get を行うシナリオを作成、実行し、提案手法による通信回数の削減効果を測った。1,000 ノードで DHT を構成し、そこに 5 万の key-value ペアを put、続いて get した。一括フォワーディングを行う際の初期 bundle サイズ (ID の数) は 10 とし、クラスタリングは、実験に先立ち、4.1 節の方法で行った。

本章の実験で put/get したキーは、ASCII バイト列 key に通し番号を連結することで生成した。具体的には key0, key1, ... となる。宛先 ID は、このバイト列を暗号的ハッシュ関数 SHA-1 に与えて出力を得、先頭から必要なビット数、Pastry なら 128 ビット、それ以外なら 160 ビットを取り出したものとなる。この方法で生成した宛先 ID は、本章の実験では、すべて異なる値となった。つまり、ここでは 5 万の key-value ペアに対して、宛先 ID もすべて異なる 5 万通りの値となった。これらの宛先 ID は、暗号的ハッシュ関数によって生成されるため、ID 空間中に偏りなく分

布する。

シナリオの内容は次のとおりである。DHT を応用した際の初期稼働時、または、全データのバックアップ (1 章) を想定して、すべての key-value ペアを put/get する。所要時間は、加入開始から 1,040 秒となる。

- (1) 1,000 ノードを起動する。
- (2) 1,000 ノードを 20 ミリ秒ごとに DHT に加入させ、オーバレイを構築する。
- (3) 10 秒間、制御を休止。
- (4) 10 ミリ秒に 1 つのペースで、5 万の key-value ペアを put。
- (5) 10 秒間、制御を休止。
- (6) 10 ミリ秒に 1 つのペースで、5 万の key-value ペアを get。

put, get を行うノードは、シナリオ生成時に乱数で選んだ。一括フォワーディングを行う場合は、100 ミリ秒ごとに 1 つの bundle を put, get することで、10 ミリ秒に 1 つのペースとした。

図 4 に、アンダーレイでの 1 秒あたりの通信回数を示す。図 4a, b, c はそれぞれ、ルーティングアルゴリズムとして Chord, Pastry, Kademlia を用いた場合の結果である。Koorde は Chord と、Tapestry は Pastry とアルゴリズムの類似点が多く、傾向が同様であったため、ここでは代表して Chord, Pastry, Kademlia の結果を示す。また、ここでのフォワーディング様式 (図 1) は反復様式であるが、再帰様式も反復様式と同様の傾向を示した。

図 5 には、put と get に要した通信回数を示す。オーバレイの構築に要する通信回数は一括フォワーディングの影響を受けないので、ここでは算入しない。値は、一括フォワーディングなしの場合を 1 とした比である。

図 4, 図 5 中の “serial” は一括フォワーディングなしの場合、“random” と “clustered” はありの場合を表す。bundle サイズ制限 (3 章) のための put, get 要求のグループ化を、“random” は無作為に行なった場合、“clustered” は宛先 ID に基づいてクラスタリング (4.1 節) した場合を表す。

“clustered” では、一括フォワーディングによって、配送に要する通信回数が Pastry で 34%、Koorde や Tapestry で 12% まで減った (図 5)。

Pastry での削減効果が他より低めなのは、Pastry では、メッセージ配信に要する通信に対して、定期的なノード間通信の割合が他よりも高いからである。このことは、図 4b の 500 秒過ぎ、put 完了後の通信回数落ち込み具合が少なめであることにも表れている。

(Kademlia の “clustered” を除き) put より get の

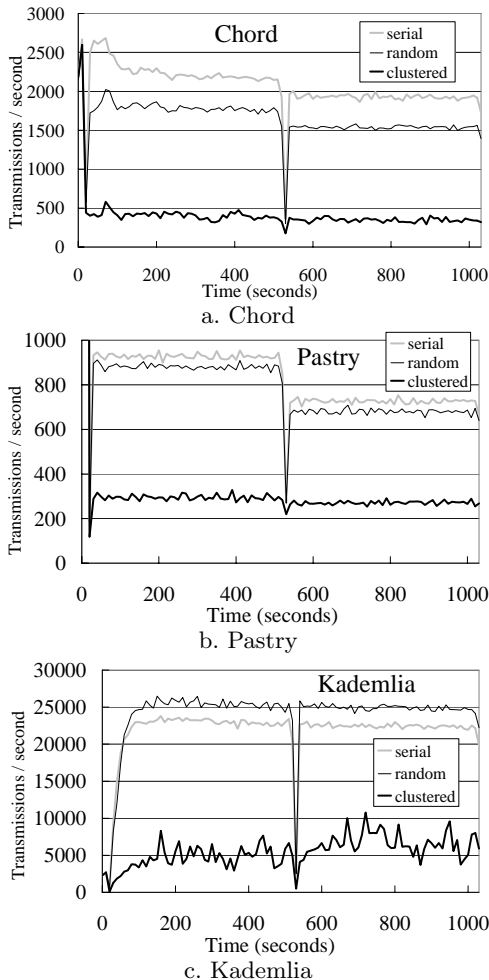


図4 オーバレイでのメッセージ配送に要したアンダーレイでの通信回数

Fig. 4 Number of transmissions on an underlay to deliver messages on an overlay.

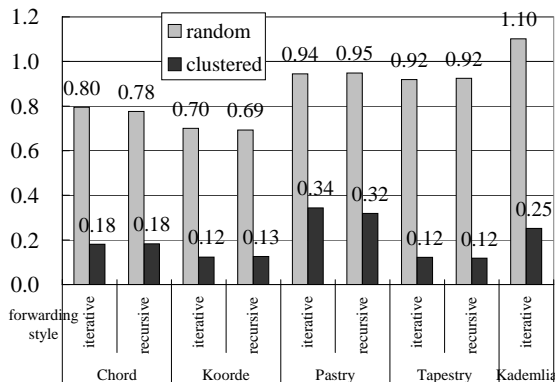


図5 一括フォワーディングを行わない場合に対する通信回数の比
Fig. 5 Ratio of the number of message transmissions with collective forwarding to the case without the technique.

ための通信回数の方が少ないことは、OWのプロトコルに起因する。getは担当ノードからの返答をもって完了するのに対し、putでは配送に対する返答の後、あらためて値も含めたput要求を送る。これによって、大きなものとなる可能性がある値を、経路に沿ってフォワードしていくことを防いでいる。この、putとgetの差は、図4ではPastryにおいて一番明確に表れている。経路が短いほどこの差は顕在化するからである。Pastryでの経路はOWが提供する他のアルゴリズムより短くなる傾向がある。

(Kademliaを除き)“random”であっても通信回数は若干減っている。これは、クラスタリングを行わなくても次ホップの偶然の一致はある割合で発生するから、と説明できる。

Kademliaでは、“serial”より“random”での通信回数が増えており(図5)、“clustered”ではput中よりget中の通信回数が多い(図4c)。これは、通信回数の多いノードは、他ノードの生存確認のためにさらに多くの通信を行う、というKademliaの経路表管理方式に起因する。

4.3 メッセージ配送の所要時間

DHTからのgetに要する時間を測った。OWの分散環境エミュレータを用いて計算機1台上に1,000ノードからなるDHTを構成し、そこから1万のkey-valueペアをgetした。分散環境を模すため、ノード間の通信に1ミリ秒の遅延を付加した。これは、LANとしては大きく、広域ネットワーク(数ミリ秒~)としては小さい、という程度の値である。

図6に所要時間を示す。図4、図5(4.2節)と同様に、“serial”は一括フォワーディングを行わない場合、“random”はput、get要求のグループ化を無作為に行った場合、“clustered”はクラスタリング(4.1節)を行った場合を表す。

並行性(concurrency)は、オーバーレイ上で同時に処理され得るメッセージ配送(つまりget)要求の数を表す。“random”、“clustered”の並行性10は、サイズ(宛先ID数)10のbundleを1つずつ配送要求することを意味する。“serial”の並行性10では、1ノードに対して10の接続を張り、それらを通じて、10までのメッセージ配送を並行して要求した。並行性100とは、10の接続を通じてサイズ10のbundleを配送要求することを指す。並行性10の“serial”は、同じだけの並行性を活かせるという観点では並行性10の一括フォワーディングとの比較が適当だと言えるが、一方で、ノードに対して10の接続を張るので、並行性100の一括フォワーディングとの比較が適当だとも

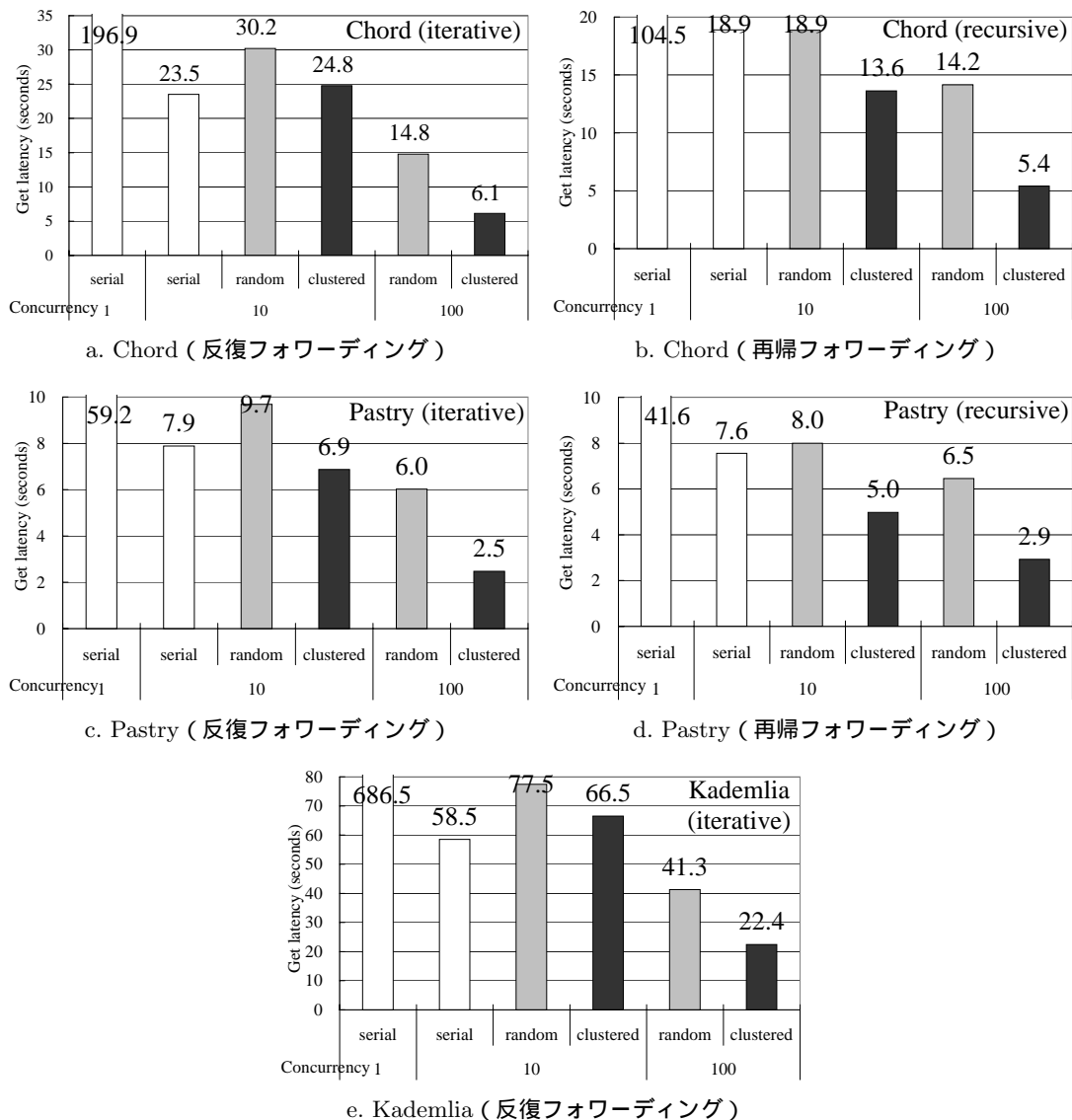


図 6 10,000 メッセージの配送に要した時間
Fig. 6 Time to deliver 10,000 messages.

考えられる。

要求メッセージ群を 10 ずつクラスタリングした場合 (“clustered”, 並行性 10), 並行性 1 の “serial” と比較して, 配送の所要時間は 13.0%(Chord, 再帰) ~ 9.7%(Kademlia) まで短縮された。これは, 一括フォワーディングなしに, 複数の接続を並行に利用した場合 (“serial”, 並行性 10) の 18.2%(Pastry, 再帰) ~ 8.5%(Kademlia) に近い短縮率である。さらに, 複数の bundle を並行して配送要求することで (並行性 100), 所要時間を 7.03% (Pastry, 再帰) ~ 3.12% (Chord, 反復) まで短縮できた。

5. 関連研究

提案手法と同様の効果を得るために, フォワーディングを行うことなく宛先 ID 群に対する担当ノードを算出して, 直接, 担当ノードに対してまとめて put, get 要求を行うという手法が考えられる。そのためには, まず, オーバレイ上のノード群について ID を把握する必要があり, そのためのコストがフォワーディングより低くなるか否かは明らかではない。特に, ノードの離脱, churn を前提とすると, コスト高となることが予想される。そもそも, 各ノードが持つ情報を

$O(\log N)$ 以下に抑えられることが構造化オーバーレイの特徴であり、この強みを捨てることとなる。提案手法は、この強みを損ねずに、多数のメッセージ配送要求を効率化する。

提案手法のように配送要求の時点で bundle 化しておくのではなく、ノード上で出会ったメッセージ群を動的に bundle 化することも考えられる。しかしそのためには、ノード上でメッセージ群の待ち合わせが必要となり、その待ち時間の分、配送が遅くなってしまふ。

Xcast⁹⁾ 対応ルータも、一括フォワーディングと同様に、次ホップが同一となる複数の宛先をまとめて扱い、その次ホップに対してパケットをただ1度フォワードする。Xcast は、メンバ数の少ないグループ向けの、IP 網上のマルチキャストプロトコルである。Xcast パケットはヘッダに複数の宛先 IP アドレスを含み、それを、Xcast 非対応ルータは宛先のうちの1つに向けて転送、対応ルータは複数の宛先について次ホップを決定して、各次ホップに対して1回ずつフォワーディングを行う。Xcast 対応ルータが提案手法と異なるのは、Xcast は IP 層のプロトコルである点、マルチキャストが目的であり、ユニキャストメッセージの一括配送は対象としていない点である。また、宛先が IP アドレスであり、よほど共通プリフィクス長が長い場合を除いて、複数の宛先に対して経路の重複具合を見積もることはできず、よって、クラスタリング(4.1節)によるフォワーディング回数の削減といった最適化の余地はない。

IP マルチキャストも、多数の宛先に向けたメッセージ配送を効率化する技法である。どのプロトコルも、事前に配送木を構築する。それに対して提案手法は、事前に受信者の確定は行わず、送信のたびに宛先 ID 群を指定できるという特徴がある(2章)。

提案手法のような複数配送要求の効率化ではなく、単一の配送を効率化するには、フォワーディング1回あたりの通信遅延を小さくするか、経路長を短くすればよい。前者の試みとして proximity routing¹⁰⁾ が、後者の試みとして例えばフォワーディングを定数回に抑える方式¹¹⁾ や1回で済ませる方式¹²⁾ がある。必ず1回で済ませる key-value ストア Dynamo¹³⁾ も同様の試みである。

6. ま と め

本論文では、オーバーレイでのメッセージ配送を効率化し、IP 網などアンダーレイの負担を軽減する手法、一括フォワーディングを提案した。次ホップが同一と

なるメッセージをまとめてフォワードすることで、総フォワーディング回数、ひいてはアンダーレイでの通信回数を減らす。

提案手法を Overlay Weaver に実装し、いくつかのルーティングアルゴリズムと組み合わせて動作させた結果、通信回数は34%~12%まで減った。所要時間は、逐次に配送した場合の13.0%~9.7%となった。さらに、複数の bundle を並行して配送要求することで、所要時間は7.03%~3.12%まで短縮できた。

今後は、DNS やセンサデータといった実データを記憶、処理するオーバーレイを構築し、提案手法の適用可能性や効果を調べていく。また、構造化オーバーレイに限らず、非構造化を含めたオーバーレイ一般への提案手法の適用も興味深い。

謝辞 提案手法の有効性について議論して頂いた中尾彰宏様、阿多信吾様、DHTでの大量データの取り扱いを考えるきっかけを下さった藤田昭人様、土井裕介様、また、本研究の基盤である Overlay Weaver を開発する機会を下さった関口智嗣様、田中良夫様に感謝致します。

参 考 文 献

- 1) Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. and Stoica, I.: Towards a Common API for Structured Peer-to-Peer Overlays, *Proc. IPTPS'03* (2003).
- 2) Pappas, V., Massey, D., Terzis, A. and Zhang, L.: A Comparative Study of the DNS Design with DHT-Based Alternatives, *Proc. INFOCOM 2006* (2006).
- 3) Azureus: Java BitTorrent Client. <http://azureus.sourceforge.net/>.
- 4) BitTorrent. <http://www.bittorrent.com/>.
- 5) Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay Weaver: An Overlay Construction Toolkit, *Computer Communications (Special Issue on Foundations of Peer-to-Peer Computing)*, Vol. 31, No. 2, pp. 402-412 (2008).
- 6) Overlay Weaver: An Overlay Construction Toolkit. <http://overlayweaver.sf.net/>.
- 7) 首藤一幸, 加藤大志, 門林雄基, 土井裕介: 構造化オーバーレイにおける反復探索と再帰探索の比較, 情報処理学会研究報告, 2006-OS-103-2 (2006).
- 8) Rhea, S., Geels, D., Roscoe, T. and Kubiatowicz, J.: Handling Churn in a DHT, *Proc. USENIX '04* (2004).
- 9) Boivie, R., Feldman, N., Imai, Y., Livens, W. and Ooms, D.: Explicit Multicast (Xcast) Concepts and Options, RFC 5058, IETF (2007).
- 10) Gummadi, K., Gummadi, R., Gribble, S., Rat-

- nasamy, S., Shenker, S. and Stoica, I.: The Impact of DHT Routing Geometry on Resilience and Proximity, *Proc. SIGCOMM 2003* (2003).
- 11) Gupta, I., Birman, K., Linga, P., Demers, A. and van Renesse, R.: Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead, *Proc. IPTPS'03* (2003).
 - 12) Gupta, A., Liskov, B. and Rodrigues, R.: Efficient Routing for Peer-to-Peer Overlays, *Proc. NSDI '04*, pp. 113–126 (2004).
 - 13) DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: Amazon's Highly Available Key-value Store, *Proc. SOSP 2007* (2007).

(平成 21 年 1 月 27 日受付)

(平成 21 年 5 月 8 日採録)



首藤 一幸 (正会員)

1996 年早稲田大学理工学部情報学
科卒業．1998 年同大学助手．2001
年同大学大学院理工学研究科情報科
学専攻博士後期課程修了．同年産業
技術総合研究所研究員．2006 年ウ
タゴエ (株) 取締役最高技術責任者．2008 年 12 月東
京工業大学准教授．博士 (情報科学)．分散システム,
プログラミング言語処理系, 情報セキュリティ等に興
味を持つ．SACSYS2006 最優秀論文賞．IPA 未踏ソフ
トウェア創造事業スーパークリエイター認定．情報処理
学会平成 18 年度論文賞．情報処理学会平成 19 年度山
下記念研究賞．日本ソフトウェア科学会, IEEE-CS,
ACM 各会員．