下位アルゴリズム中立なDHT実装への耐churn手法の実装

首　藤　一　幸†

ノードの頻繁な離脱と加入，つまりchurnに対する耐性向上は，分散ハッシュ表(DHT)の大きな課題である．本論文では，いくつかの耐churn手法とその効果を示す．ここで示す手法はどれもDHT層に対する実装であり，その下のルーティング層への変更は必要としない．ゆえに，特定のルーティングアルゴリズムに依存せず，様々なアルゴリズムと組み合わせて用いることができる．耐churn手法のどれをどういうパラメータで組み合わせて用いるべきかは応用に依存し，一意には定まらない．適切な手法とパラメータを見出す方法を考察する．

Churn Resilience Improvement Techniques in an Algorithm-neutral DHT

Churn resilience is an important topic in DHT research. In this paper, I present techniques to improve churn resilience and effect of them. All the techniques can be implemented in a DHT layer and require no change to an underlying routing layer. In other words, they do no depend on a specific routing algorithm and can work with various algorithms. They are dependent on a DHT application which techniques and what parameters are optimal. I last discuss how we can determine it.

1. はじめに

分散ハッシュ表(DHT)とは，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でハッシュ表の機能を達成する仕掛けである．ハッシュ表という汎用性の高い機能を提供するため応用の範囲が広く，様々な名前解決に用いることができる．例えばDNSへの応用が盛んに研究されている[1)]．

ネットワークサービスを提供する側にとって，非集中分散(peer-to-peer)システムを用いたサービス提供には，次のメリットがある．

- 低い管理・提供コスト
- 高いスケーラビリティ
- 高い信頼性

ここで，信頼性確保のために，少数のサーバを用いる場合と同様に個々のサーバの信頼性を高めるというアプローチを採ってしまうと，管理・提供コストがサービス提供側の台数に比例して高くなってしまい，元の木阿弥である．逆に，サービス受益者，利用者のノードをもサービス提供に用いる場合は，そもそも各ノードの信頼性，可用性はまったく期待できない．

† ウタゴエ(株)
Utagoe Inc.

そこで，非集中分散システムには，サービス提供ノードの離脱や加入（故障や復帰）がある程度の頻度で起こることを前提として，それでもサービスを継続できることが求められる．頻繁なノードの出入り，つまりchurnへの耐性は，DHTにおいても重要な課題である．

本論文では，DHTを対象としたいくつかの耐churn手法，および，その効果を示す．本研究の特色は，ここで示す手法のどれもが，下位のルーティングアルゴリズムに依存しない点である．どの手法もDHT層に対して実装するものであり，その下のルーティング層への変更は必要としない．これにより，どの手法も，Chord，Kademlia等，それぞれ特性の異なる様々なルーティングアルゴリズムと組み合わせて用いることができる．

各手法は構造化オーバレイ構築ツールキットOverlay Weaver[2),3)]に実装した．4章では各手法の効果を示す．

どの耐churn手法をどういったパラメータで使い，どう組み合わせて用いるべきかは，一意には定まらない．DHTの応用に依存する．本論文では最後に，それを見出す方法を考察する．

2. churn問題

DHTは構造化オーバレイ（structured overlay）の

第19回 コンピュータシステム・シンポジウム (ComSys 2007) 論文集,
pp.191-198, 2007年 11月

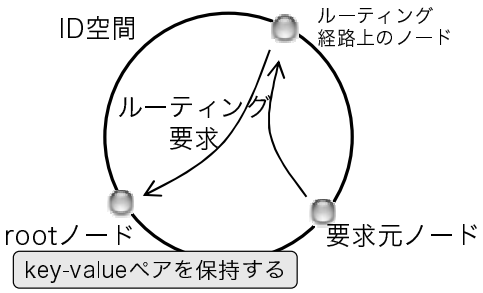


図 1 DHT での put,get のためのルーティング
Fig. 1 Routing to put and get to a DHT.

一応用であるという整理[4] に従うと，DHT の put, get 処理は次のように説明できる（図 1）.

put(key,value) key をキーとしてルーティングを行うと，何ノードかを経由して，key を担当するノード，すなわち root ノードに到達する．そのノードに，key-value ペアを保持させる．

get(key) key をキーとしてルーティングを行うと root ノードに到達する．そのノードから，key と結びつけられている value を受け取る．

ここで，DHT を構成するノードの離脱や加入があると，put したはずの値を get できないという問題が起こり得る．get 失敗の原因は，より直接的には以下の通りである．

(1) put 後に（root ノード等の離脱により）key-value ペアが消滅した．
(2) root ノードが key-value ペアを保持していない．
  (a) put が済んだ後で加入したノードが新たな root になった．
  (b) 経路表が不完全といった理由で，put 時のルーティングが root ノードに到達しなかった．
(3) ルーティング経路上のノードが要求の転送中に離脱した．(recursive ルーティング[5] でのみ発生する．)

例えば，key-value ペアを保持している root ノードが離脱してしまえば，その値は get しようがない．しかし，put 後に root ノードが離脱してしまった場合でも，別のノードに複製を作っておき，get でそれら複製の値を得られるならば，get を成功させることはできる．3 章で，こういった耐 churn 手法をいくつか示す．

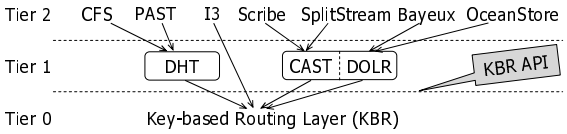


図 2 Key-based routing (KBR) (Dabek et al.[4] の Figure 1)
Fig. 2 Key-based routing (KBR) (Figure 1 from Dabek et

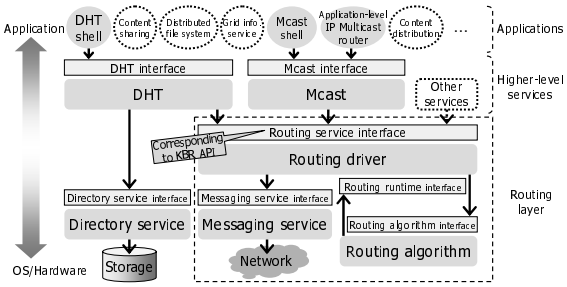


図 3 Overlay Weaver の構造
Fig. 3 Components organizing runtime of Overlay Weaver.

3. 耐 churn 手法

ここでは，耐 churn 手法を述べる．各手法は，構造化オーバレイ構築ツールキット Overlay Weaver[2),3)] に実装し，その効果を測定した．

Overlay Weaver は，Dabek らの抽象化（図 2）[4] に従い，ルーティング層の上に DHT やマルチキャスト等の高レベルサービス層が載るという構造を採っている（図 3）．この構造には，ルーティング層と高レベルサービスを自由に組み合わせられるという利点がある．例えば，同一の DHT 実装に対して，様々なルーティング層の実装やアルゴリズムを組み合わせられるということである．

そこで，耐 churn 手法が DHT 層だけに対する実装であるならば，ルーティングアルゴリズムに依存することなく，様々なアルゴリズムと組み合わせて用いることが可能となる．これは当然であるように聞こえるが，そうでもない．耐 churn 手法は他のノードとの通信を伴うため，他ノードの情報を得るために，ルーティングアルゴリズム固有の表（例えば Pastry の leaf set）を参照するものも多い．耐 churn 手法をアルゴリズムに依存させないためには，ルーティング層からはアルゴリズム中立な API を提供するようにし，ルーティング層内部への直接アクセスは注意深く避けねばならない．

3.1 ルーティング層の API

Dabek らの抽象化でも，DHT 層に耐 churn 手

法として複製機能を実装できるよう，配慮がなされている．ルーティング層が上位層に対して手続き `replicaSet(key,max_rank)` を提供しているのはそのためである．この手続きは，key に対する複製を作成すべきノードの順序付きリストを返す．返されるリストは，key に対する root として適切な順にノードを並べたものである．つまり，先頭のノードが離脱した場合には次のノードが root となり，そのノードが離脱した場合にはまた次のノードが root となる，というリストである．

今回，Overlay Weaver に，これと同じ意味を持つ手続きを用意した．ただし，複製以外の目的にも用いるため，replica という語を避け，root 候補のリストという意味を反映した名前を付けた：`rootCandidates(ID target,int maxNumber)`．

以下，次の 4 手法を述べる．このうち上 3 つは，手続き `rootCandidates` を用いて実装した．

- 複製
- 加入時委譲
- 複数 get
- 自動再 put

3.2 複　　製

put 時，key を担当する root ノードだけでなく，他のいくつかのノードに key-value ペアを保持させる．これにより，put 後に root ノードが離脱した場合でも，get 時には他のノードから値を取得できる．複製は，root 候補としての順位に沿って，いくつかのノードに保持させる．つまり `rootCandidates` 手続きで得られるリストに含まれるノードに保持させる．こうすることで，put 時に root だったノードが離脱した場合でも，ルーティングの結果，次に root として適切なノードに到達し，そのノードは複製を保持しているため get は成功する．

root 候補リストではなく，何らかの手順で key から得られる複数の値をキーとして複製を put しておくという方法も考えられる．例えば，key の先頭 2 ビットを変化させて 4 通りの値を作ればよい：$key \oplus$010..(2 進)，$key \oplus$ 100..，$key \oplus$ 110... しかしこの方法には，通信の回数が増すという難点があるため，今回は採用していない．root 候補リストへの複製であれば，1 度のルーティングと複製数だけの put 要求で済むところ，この方法では複製の数と同じ回数のルーティングが必要となる．

今回の実装では，root 候補ノード群への put 要求は，put 要求元ノード，root ノードのどちらが行うこともできる．どちらが行っても通信の回数は同一だが，ノード ID の数値とネットワーク近接性の間に何か関係がある場合は，効率に差が出る．例えば proximity identifier selection (PIS)[6] を採用していてノード ID の数値的な近さがネットワーク近接性を反映しているなら，root ノードが要求を行った方が効率がよい．今回の実装は近接性を考慮していないため，差は出ない．

複製に関するパラメータは次の 2 つである．

- 複製数
- 複製の put 要求を put 要求元ノードと root ノードのどちらが行うか

複製数が 1 の場合，root ノードにだけ key-value ペアを保持させる．

複製とキャッシュの違いには注意されたい．キャッシュは，get/put 時に key-value ペアを取り扱うノードが性能向上を目的としてその値を保持しておくというものである．偶然，churn 耐性向上に寄与することはあっても，耐 churn を狙った手法ではない．とはいえ，キャッシュを複製として活用することも考えられる．

3.2.1 複製群の一貫性

churn 状況下で複製を作成，維持するので，同一の key に対する value がノードによって異なるという状況が懸念される．上書きされるべき古い value が残ってしまう場合や，複数ノードからの put のタイミングが近く競合状態が起きるといった場合である．

しかしこれは，本研究が前提とする Overlay Weaver の DHT 実装では問題とならない．当該 DHT 実装では Bamboo[7] と同様に，同一の key に対して複数の value が結び付くからである．つまり，通常のハッシュ表で起きるような，同一 key に対する複数 value の競合は起きず，get では，put された value すべてが得られる．

それでも，DHT に対する remove 操作で削除したはずの value が得られてしまうということは起き得る．remove の処理は基本的に put と同じであり，put では key-value ペアを保持させる代わりに，remove では削除する．remove が失敗する原因は put とまったく同じなので，本研究では代表して put に続く get の成功率を調べる（4 章）．

ある key に対応する value がただ 1 つとなる DHT では，churn 状況下での完全な一貫性維持が困難である以上，例えば多数決といった get 時の対策が必要となろう．

3.3 加入時委譲

DHT に新たに加入してきたノードに対して，そのノードが root となるような key-value ペアを他のノー

表 1 各耐 churn 手法が対象とする get 失敗原因
Table 1 Causes of get failures each technique treats

	複製	加入時委譲	複数 get	自動再 put
key-value ペアが消滅	√			√ (複製が前提)
root が key-value ペアを保持せず:				
新ノードが root に		√	√	√
put 時 root 不到達			√	√
経路上ノードが離脱				

ドから委譲する．これにより，put 後に加入したノードが root となるような get 要求にも答えられるようになる．

今回の実装では，新規加入ノードの方から，自身が担当すべき key-value ペアを保持していそうなノードいくつかに対して問い合わせを行う．具体的には，`rootCandidates` で得たリストの先頭から順に，いくつかのノードに問い合わせる．問い合わせを受けたノードは，新規加入ノードの方が root として適切であるような key-value ペアを返す．新規加入ノードは返された key-value ペアを保持する．この受け渡しの際，元から保持していた方を削除するわけではないので，委譲というより実際はコピーを行っている．

加入時委譲のパラメータは，新規加入ノードからいくつの root 候補ノードに対して問い合わせるか，である．

3.4 複 数 get

get 時，root ノードに対してだけでなく，いくつかのノードに対して key に対応する key-value ペアを問い合わせ，要求する．具体的には，`rootCandidates` で得られる root 候補リストに含まれるノードに対して問い合わせる．これにより，put 後に DHT に加入したノードが root となった場合でも，新ノード加入前の root に対しても get を要求するので，key-value ペアを取得できる場合がある．

put 後に加入したノードが root になるというこの状況は，前述の加入時委譲でも救うことができる．どちらの手法でも救えるということは，両手法の効果は重複するということである．

複数 get は，何らかの理由で，put 時のルーティングが最適な root ノードではなく次善のノードに到達してしまった場合にも有効である．ルーティングアルゴリズムによっては，経路表が完全でない場合に root ノードに到達しないことは起こり得る．この場合，次善のノードが key-value ペアを保持してしまうが，この手法によって，次善のノードに対しても get を要求できることがある．

複数 get のパラメータは，get 要求先ノード数である．これが 1 の場合，root ノードにだけ要求する．

3.5 自動再 put

各ノードが，自身が保持している key-value ペアを DHT 上に put する．通常の put と同様に複製も作る．この自動再 put は，通常の put，get のタイミングとは関係なく，平常時に，各ノードが自律的に行う．

key-value ペアは，加入時委譲を行うことで若干の補充はなされるものの，ノードの離脱によって減っていく．自動再 put はこの補充を狙った処理である．

自動再 put には，ある程度，加入時委譲と同様の効果もある．つまり，自動再 put によって，put 後に加入したより適切な root ノードに対して key-value ペアが渡される．ただし，自動再 put 処理には時間間隔があるため，加入時委譲や複数 get とは異なり，自動再 put が起こるより前の get 要求は救えない．

複製数が 1，つまり root ノードだけが key-value ペアを保持する場合，自動再 put には意味がないように見えるが，それでも加入時委譲と同様の効果はある．

自動再 put のパラメータは，時間間隔である．具体的には，自ノードが保持している key-value 群を再 put する処理と処理の間に，指定した時間だけの休止時間をとる．また，複数ノードの自動再 put 処理が同期してしまわないよう，休止時間は乱数で増減させている．

churn 耐性向上 4 手法のうち，この自動再 put だけが手続き `rootCandidates` （3.1 節）を使わない．

3.6 各手法が対象とする get 失敗の原因

表 1 に，それぞれの耐 churn 手法が，2 章で述べた get 失敗原因のどれを対象としたものかを整理する．

複製は，ノード離脱に伴って DHT から key-value ペアが消滅してしまうことを防ぐ．自動再 put は，ノード離脱に伴う複製数の減少を補充する．ただし，複製数が 1 の場合，この補充効果はない．

加入時委譲と複数 get は，root ノードそのものが key-value ペアを保持していない，という状況への対策である．加入時委譲はその状況を防ぎ，複数 get はその状況でも key-value ペアを取得できるようにする．ただし，経路表が完全でないなどの理由で put 時に

root ノードに到達できなかった，という状況は，加入時委譲では救えない．複数 get では救える可能性がある．

root ノードが key-value ペアを保持しないという状況は，自動再 put でも救える場合がある（3.5 節）．

recursive ルーティングの最中に経路上のノードが離脱するとルーティングを完遂できないという get 失敗原因に対しては，今回は解決を与えていない．複数の問い合わせを並行して送出するといった対策があり得る．

4. 各手法の効果

3 章で示した耐 churn 手法の効果を計測した．計算機 1 台の上で 1000 ノードを動作させ，churn を発生させて，DHT の get が成功した回数を数えた．

実験には，Overlay Weaver が提供する分散環境エミュレータを用いた．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，通信部分を除き，実環境で動作するものと同一のものである．

このエミュレータが提供する通信層は，送信されたメッセージを，計算機の性能が許す限りの速さで宛先ノードに届ける．ここで，メッセージのコピーは行われないため，帯域幅は無限大となっている．つまり，LAN やインターネットなど物理的な通信媒体に起因する帯域幅の制限や通信遅延がない，理想的な通信環境を模していることになる．churn による get 失敗はノードの離脱と加入という通信環境とは無関係の原因によって起こる．このため，本実験の結果，つまり get 成功率への通信環境の影響は小さいと考える．

4.1 条　　件

今回は，ノードの離脱と加入が頻繁に起こる churn のシナリオを生成して使用した．すべての実験で同一のシナリオを用いた．

シナリオの内容は次の通りである．

(1) 1000 ノードを起動する．
(2) 1000 ノードを 0.15 秒毎に DHT に加入させる．
(3) 1000 通りの key-value ペアを 0.2 秒毎に put する．
(4) 1000 通りの key-value ペアを 0.2 秒毎に get する．

put，get を行うノードは，シナリオ生成時に乱数で選んだ．

churn は，put 開始時から get 終了時の間，起こし続けた．ノードを離脱させた直後に別のノードを加入させることで，DHT に加入しているノード数を常に 1000 に保つ．ノード数を保つというこの churn モデルは，Rhea らの実験[8] と同じものである．加藤らの churn モデル[9),10)] はこれとは異なり，離脱直後に加入を起こさないため，ノード数が変動する．

churn の頻度は平均 2 ノード/秒とし，ポアソン分布に従って発生させた．ノードの平均生存時間は 1000(ノード)/2(ノード/秒) = 500 秒 となる．

通信のタイムアウトは 3 秒，ルーティングのタイムアウトは 10 秒と設定した．つまり，1 度のルーティング（put や get）のうちに 4 回，離脱済みのノードへの通信を試みると，確実にルーティングがタイムアウトする．通信のタイムアウトが起きた場合には，経路表から通信相手のノードを削除する．

実験には 2.8 GHz Pentium D プロセッサ，x86-64 用 Linux 2.6.21，x86 用 Java 2 SE 5.0 Update 12 の HotSpot Server VM を用いた．Overlay Weaver の版は 0.6.4 である．すべての実験は 6 回行い，最良値と最悪値を捨て，残り 4 回分の値を平均したものを結果として採用した．

4.2 結　　果

図 4 に結果を示す．実験は，Overlay Weaver が提供するすべてのルーティングアルゴリズムと，iterative / recursive ルーティング[5)] の全組み合わせに対して行った．ここでは，Chord，Pastry，Kademlia を iterative ルーティングで用いた際の結果を示す．

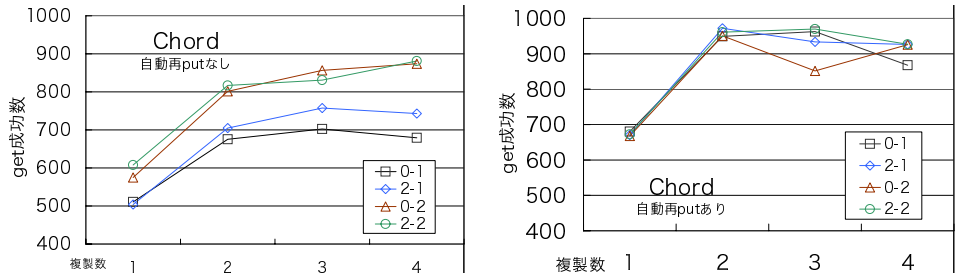
グラフの縦軸が，get 1000 回のうち成功した回数を示す．この値が 1000 に近いほど良い結果である．横軸は，複製の数 (1〜4) を表す．各アルゴリズムについて，横に 2 つのグラフが並んでいる．左は自動再 put なし，右は自動再 put ありの結果である．自動再 put の間隔は，平均 30 秒とした．

グラフ中の 4 本の線は，それぞれ，加入時委譲と複数 get のパラメータが異なる．加入時委譲のパラメータは，新規加入ノードから問い合わせる先のノード数であり，0（加入時委譲なし）または 2 とした．複数 get のパラメータは，get 要求先ノードの数であり，1（複数 get なし）または 2 とした．グラフ中に “数字-数字” とあるのは，両パラメータを表しており，“加入時委譲-複数 get” という順に並んでいる．

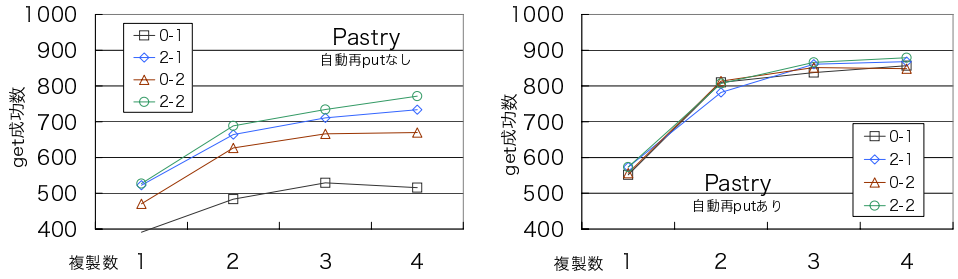
4.3 考　　察

図 4 のグラフから次を読み取ることができる．

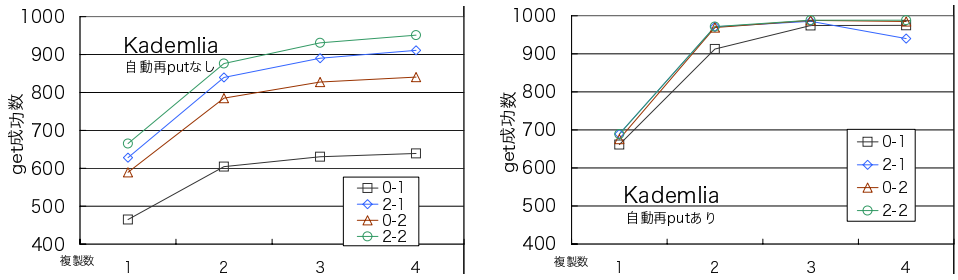
- 複製数が多いほど，get 成功率が上がっている．
- 加入時委譲と複数 get は，どちらも，行った方が get 成功率が上がっている．
- 自動再 put なしの場合，Pastry と Kademlia で



(a) Chord



(b) Pastry



(c) Kademlia

図 4 各手法の効果
Fig. 4 Results of proposed techniques.

は，複数 get（get 要求先:2）よりも加入時委譲（問い合わせ先:2）の方が効果が高かった．Chord では逆に，複数 get の方が効果が高かった．

- 自動再 put を行うと，複数 get や加入時委譲による差がほとんど見られなくなった．
- 複製数 3 よりも 4 の方が結果が悪い場合がある．

この傾向は，recursive ルーティングでもまったく同様であった．また，Tapestry の傾向は，アルゴリズム上の類似点が多い Pastry と同様であった．

なお，耐 churn 手法の適用にもかかわらず get の失敗が起きているのは，次の原因によるものであった．

- 耐 churn 手法が追いつかなかった．
  複製を保持する全ノードの離脱，新たな root に対する自動再 put より早い get，など．
- 新たに加入したノードの経路表が不完全で，適切な root に到達しなかった．
- タイムアウトが多数回起き，ルーティングを完遂できなかった（4.1 節）．

これらの結果はアルゴリズム間の優劣を表すものではない点に注意されたい．各アルゴリズムには固有のパラメータ（例:Chord の `stabilize` 間隔）があり，中には churn 耐性に影響を与えるものもある．今回は，Overlay Weaver 0.6.4 の既定のパラメータを用いた．

ここまでで，今回示した各手法の効果を確認できた．しかし，各手法を採用するか否か，またそのパラメータを決めるためには，効果だけでなく，費用対効果を調べる必要がある．次章では，費用対効果を考察する．

表 2 各耐 churn 手法の処理（通信）タイミング
Table 2 DHT processes in which each technique is involved

	加入	put	get	平常時
複製		√		
加入時委譲	√			
複数 get			√	
自動再 put				√

5. 費用対効果

本章では，耐 churn 手法の費用対効果を算出する方法を考察する．

まず，効果は get 成功率として表れるので，定量的に表すとしたら，get 成功率の関数とすることが適切だと考える．

続いて，費用について検討する．耐 churn 手法の適用によって増加し得る費用には，次がある．

- 通信量
- 加入，put，get に要する時間
- メモリやストレージの消費量
- プロセッサの処理

従来研究[8),9)] は，通信量と get に要する時間（lookup latency）に着目してきた．他の費用が重視されていないのは，現在の PC や LAN，インターネットで動作する DHT を想定すると，それらはボトルネックとならないからであると推測する．

とはいえ，複製数を n とすると，1 とした場合と比較して n 倍のストレージ（かメモリ）を消費する．データ量が多い場合，組み込み用途などで容量が限られる場合には，費用として充分に高価なものとなり得る．つまり，費用を考える際には，応用の環境を想定することが欠かせない．

応用ごとに異なる，系の振る舞いも考慮する必要がある．表 2 に，耐 churn 手法それぞれについて，処理および通信が起きるタイミングを示す．この表から判ることは，put の回数が多いほど複製の費用は高くなり，get の回数が多いほど複数 get の費用は高くなるということである．また，自動再 put では，put されている key-value ペアの数に応じた通信が継続的に起き続ける．つまり，加入，put，get が起きずとも費用がかかり続ける．

ここで例えば，DNS のような，加入や put より get の頻度がはるかに高いような応用を想定した場合，複数 get の通信費用は高くつくが，複製と加入時委譲の費用は相対的に安くあがる．すると，頻度の低い手法については，例えば複製数を増やすといった，1 回あたりの費用が高くつくパラメータ設定が可能となる．一方で，センサデータの保持といった，get と比較して put もそれなりに起きるような応用であれば，複製の通信費用も無視できない．

このように，耐 churn 手法の費用を算出するためには，加入，put，get それぞれの頻度を何かしら仮定する必要がある．定量的な算出には，key-value ペアの値やサイズも含めて，応用に則したシナリオや，現実のトレースデータを用いた実験が必要となる．

4 章の実験で用いたシナリオ（4.1 節）は，加入，put，get の回数が同じという人工的なものであり，この実験について通信量を調べることにはあまり意味がない．

6. 関 連 研 究

Dabek らの階層モデル（図 2）[4)] も，複製機能を実装するための API を定義している（3.1 節）．そこでは，階層モデルおよび層間の API が提案，考察されており，実証は伴っていない．

本論文では，その API を用いつつ，手続きの種類を増やすことなく，複製に加えて，加入時委譲，複数 get，自動再 put といったいくつかの耐 churn 手法を実装できることを示した．また，4 章では，様々なルーティングアルゴリズムと組み合わせた場合の各手法の効果を示した．

Rhea らは，彼らの DHT 実装 Bamboo[7)] に実装した耐 churn 手法を評価した[8)]．実験には，ネットワークエミュレータ ModelNet を用い，40 台の PC で 1000 ノードを動作させている．経路表中に残る離脱ノードの情報を，能動的かつ定期的に検出するか，通信失敗をもって受動的に検出するかを比較している．また，TCP にならった通信タイムアウト時間の調整や proximity neighbor selection (PNS)[6)] の効果を，get に要する時間という形で計測し，評価している．

それらの手法は，Dabek らの階層モデルで言うと，DHT 層ではなく，その下のルーティング層に実装する手法である☆．それに対し，本論文で述べた各手法は，DHT 層が対象である．Rhea らの手法[8)] は，Bamboo の実装や Pastry 由来のアルゴリズムを前提としているのに対し，本論文の各手法は，DHT 層への実装であるため，自然と，様々なルーティングアルゴリズムと組み合わせて用いることができる．

本論文の手法と Rhea らの手法は，対象とする層が異なるため，競合はせず，組み合わせて用いることができる．実際に，Overlay Weaver の通信層は，実環

☆ Bamboo はそもそもこの階層モデルを採っていない．

境では，彼らの提案する TCP スタイルのタイムアウト時間の調整手法を採用している．

加藤らは，彼らが開発したネットワークエミュレータ peeremu を用いて，いくつかの DHT 実装を評価した[9),10)]．評価対象は，Bamboo，Chord，Accordion，FreePastry であり，十数台の PC を用いて最大で 1000 ノード規模の実験を行っている．churn 時の get 成功率と，get に要する時間を計測している．

そもそも churn 耐性を高めずに済ませるスーパーノードというアプローチもある．通常のノードの中から，性能やネットワーク帯域幅，それまでの稼働時間などに応じて，強力かつ安定稼働しそうなノードをスーパーノードとして選出する．DHT（オーバレイ）の構築，維持はスーパーノードだけが行い，通常のノードはスーパーノードからサービスを受ける．

これにより，DHT に要求される churn 耐性をある程度下げることができる．また，伝統的な DHT アルゴリズムは，全ノードが双方向に通信できることを前提とするので，NA(P)T などの理由で双方向通信が困難なノードはスーパーノードとしない，つまり DHT に加入させないことで，DHT の利用が容易になるという利点もある．しかしそれでも，ノードの離脱と加入を想定しないわけにはいかない．

7. ま と め

本論文では，DHT を対象としたいくつかの耐 churn 手法と，それらの効果を示した．各手法は，Dabek らの階層モデルで言うと DHT 層に対して実装するものであり，そのため，様々なルーティングアルゴリズムと組み合わせて用いることができる．各手法を Overlay Weaver に実装し，いくつかのアルゴリズムとともに動作させ，Chord，Pastry，Kademlia での各手法の効果を示した（4 章）．

続いて，各手法の採否やパラメータを決める際に欠かせない費用対効果を算出する方法を考察した．効果は 4 章に示したが，費用は，系の振る舞い，具体的には，加入，put，get の頻度に大きく依存するため，何かしらの応用を想定してエミュレーションシナリオを作成する必要がある，という結論を得た．

今後は，DNS，センサネットワークなど，具体的な応用を想定し，効果に加えて費用も計測し，耐 churn 手法の採否やパラメータ設定の方法論を確立していく．

謝辞　日頃から議論させて頂いている加藤大志氏，門林雄基氏，土井祐介氏，藤田昭人氏，吉田幹氏，WIDE プロジェクト IDEON ワーキンググループの諸氏に深く感謝致します．

参 考 文 献

1) Pappas, V., Massey, D., Terzis, A. and Zhang, L.: A Comparative Study of the DNS Design with DHT-Based Alternatives, *Proc. INFOCOM 2006* (2006).
2) 首藤一幸，田中良夫，関口智嗣：オーバレイ構築ツールキット Overlay Weaver，情報処理学会論文誌: コンピューティングシステム， Vol.47, No.ACS 15 (2006).
3) Overlay Weaver: An Overlay Construction Toolkit. http://overlayweaver.sf.net/.
4) 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).
5) 首藤一幸，加藤大志，門林雄基，土井裕介：構造化オーバ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).
6) Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S. and Stoica, I.: The Impact of DHT Routing Geometry on Resilience and Proximity, *Proc. SIGCOMM 2003* (2003).
7) The Bamboo Distributed Hash Table. http://www.bamboo-dht.org/.
8) Rhea, S., Geels, D., Roscoe, T. and Kubiatowicz, J.: Handling Churn in a DHT, *Proc. USENIX '04* (2004).
9) Kato, D. and Kamiya, T.: Evaluating DHT Implementations in Complex Environments by Network Emulator, *Proc. IPTPS 2007* (2007).
10) 加藤大志，神谷俊之：ネットワークエミュレータによる大規模 DHT 性能評価手法の提案，分散システム/インターネット運用技術シンポジウム 2006 (2006).