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

加藤大志　神谷俊之
ＮＥＣ インターネットシステム研究所

P2P環境で疑似的なデータベース機能を提供するDHTの評価法として、ネットワークエミュレータによる評価法を提案する。エミュレーション評価を行うことによりアルゴリズムのみではなく実装を評価することができ、通信帯域や応答時間などを実際の環境に近い状態で測定できる。このような指標が得られることは、アプリケーションを開発する際には重要である。我々が開発した評価基盤を使って、4つの既存のDHT実装について評価軸を変えた複数の評価を行い性能を分析した。実験の結果、エミュレーション評価によってのみ得られるDHT実装の差が発見でき、我々の手法の有効性が示された。

A proposal of an evaluation method of large-scale distributed hash tables by a network emulator

Daishi KATO　Toshiyuki KAMIYA
Internet Systems Research Laboratories, NEC Corporation

We propose an evaluation method of large-scale distributed hash tables (DHTs) by a network emulator. The emulator allows us to evaluate not only the DHT algorithms but also the DHT implementations. Evaluating DHT implementations is important for developing DHT applications, since it essentially indicates how to use the implementations in the applications. We have evaluated four implementations and showed how different their performance are. Our method enables to evaluate the performance of DHT implementations in a practical and complicated environment. The evaluation result shows that our method is useful for evaluating DHT implementations.

1　はじめに

P2P (Peer-to-Peer)ネットワークの分野では、スケーラブルで効率のよいデータベースとしてDHT (Distributed Hash Table, 分散ハッシュテーブル)技術が注目されている。DHTはすべて対等なノードで構成されたネットワークで仮想的なデータベース機能を提供するもので、その応用としてはファイルストレージ[1]やファイル配信[4]など、様々な大規模分散アプリケーションが考えられる。

過去の研究において、DHTの様々なアルゴリズムとその実装が開発されてきた。実装が公開されているDHTとしては、例えば、Chord [11], Pastry [10], Kademlia [7]などがある。これらのDHTアルゴリズムが提供する基本機能にはあまり差がなく、性能の差は通信効率や処理応答性の優劣で比較することになる。そこでDHT性能の評価は、異なるDHTを同一の条件で動かし、その際の通信量や応答時間を比較することで行うことができる。従来の評価手法は、DHTアルゴリズムをシミュレーションによって評価する手法が主であったが、我々はDHTの実装をエミュレーションによって評価する手法を提案する。さらに、エミュレーションを実現するために我々が開発した評価基盤を用いて、複数のDHT実装を複数の評価軸で評価した結果について述べる。

本稿の構成を以下に示す。まず、2節でDHTの概要を説明し、3節でエミュレーション評価の必要性を述べる。次に、4節で評価基盤の説明を行う。5節で評価基盤を用いた評価と考察を行い、6節でエミュレーション評価の有効性を確認する。最後に、7節でまとめる。

2　DHTの概要

DHTは、すべてのノードが任意にネットワークに参加(join)・離脱(leave)する環境において、ノードのみで可用性のあるデータベース機能を提供する技術であ

る。ノードが任意に参加離脱すると可用性を100%達成することは不可能であるが、アプリケーションでカバーされることを仮定し、ある程度(例えば90%)を超えれば可用性があるものとする。我々が想定するDHTにおけるデータベースの機能は次のようなAPIで定義される。

$$put(key, value) \rightarrow void$$
$$get(key) \rightarrow \text{listof } value$$

*put*はデータを登録するAPIで*key*と*value*を指定する。*key*と*value*はどちらも文字列であり、実装によってはそのサイズが制限されることもある。*put*の戻り値はなく、登録が成功したかを知ることはできないものとする。*get*はデータを取得するAPIで*key*を指定する。*get*の戻り値はすでに登録された*value*のリストである。実装によっては、戻り値はリストではなく、最後に登録された1つのみであることもある。

3 エミュレーションによる評価の必要性

DHTを利用して大規模分散アプリケーションを開発する場合に重要となるのは、DHTの性能である。例えば、DHTがどの程度の通信帯域を消費するのか、データの取得にはどの程度の時間がかかるのか、データのサイズや量により性能はどう変わるのか、などである。DHTの性能が分からないと、アプリケーションがどの程度DHTに依存してよいか分からずアプリケーションの設計が困難になる。

これはWebアプリケーション開発におけるWebサーバの負荷テストと同様の問題である。ところがDHTの場合には、ネットワークの環境や状態、各ノードの処理能力のばらつき、負荷のパターンなどの複雑な環境に性能が大きく影響され、単純なモデル化、性能の見積りが困難である。このため、従来のDHTの研究では、複雑な環境をシミュレーションで再現して評価することが多かった。例えば、Chordのシミュレータである Chord simulatorや、複数のアルゴリズムをサポートしたp2psim [5]などがある。しかし、シミュレーション評価は実際に実装されているコードとは違う疑似的なコードで動かすため、動作の正確な再現はできないという制約がある。

これに対処する1つの方法は、MACEDON [9]で行われているように特殊な文法でコードを記述する方法である。MACEDONの枠組で記述されたコードはシミュレーション評価することができ、かつ、そのコードがそのまま実装コードとしても利用できる。しかし、MACEDONの枠組を用いない任意のプログラミング言語で記述されたDHT実装を評価することはできないため、既存の実装を比較するのには適さない。

平均起動ノード数	316
試験時間	1時間
初期join間隔	600ミリ秒
put間隔	20秒
get間隔	20秒
put回数	10
パケット遅延	kingdata [3]
パケットロス	一率0.1%
平均join時間	60分
平均leave時間	20分

表1: 標準シナリオパラメータ

これに対処する方法としては、エミュレーションによる評価がある。エミュレーションでは、想定するネットワーク環境を再現し、DHT実装に手を加えずに動作させることができる。エミュレーションによる評価は理想的な方法であるが、すべてのノードが独立して動作するというP2Pの特性を再現するには手間がかかるため、実用的に使われた例はない。我々は、複数のDHTに共通の処理を基盤として実現することで、エミュレーションによる評価を可能にした。以降の節では、我々が開発した評価基盤の説明を行い、その評価基盤を用いた実験結果を分析することで、エミュレーションによる評価の有効性を示す。

4 DHT性能評価基盤

我々が開発した評価基盤[12]は、DHTネットワークを実験室環境においてエミュレーションにより再現することができ、様々な環境を想定した実験を行うことができる。以下では、この基盤の主要な3つの機能を説明する。

1つめの機能はノードのエミュレーションで、1台の物理的なPCで複数のノードを動作させる機能である。各ノードは単一もしくは複数のプロセス(もしくはスレッド)として動作する。各ノードにはIP Aliasingを用いて個別のIPアドレスを割り当てることにより、DHT実装としては特にエミュレーションを意識することのないようにしている。

2つめの機能はネットワークのエミュレーションで、

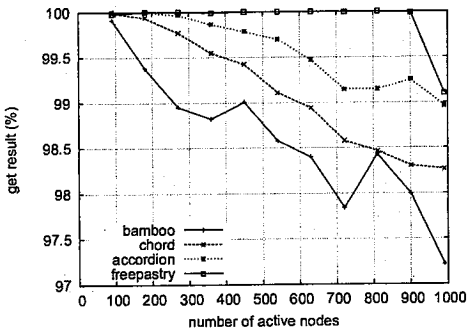

図 1: 基本実験: get 成功率

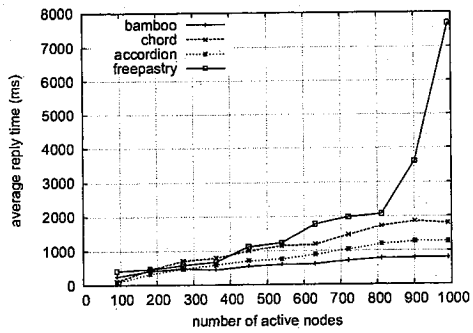

図 2: 基本実験: get 応答時間

LAN で接続された PC でエミュレーション動作する任意のノード間の通信にパケット遅延とパケットロスを発生させるものである。このネットワークのエミュレーション方式は、ネットワークのグラフを考慮するものではなく、固定的もしくは確率的な遅延やロスのモデルを再現するものである。

3 つめの機能は評価のシナリオ化で、評価試験を開始する前にすべてのノードの動作をシナリオとして記述するものである。これにより、異なる DHT を同一のシナリオで評価したり、同一のシナリオで再現試験をしたりできる。シナリオを生成するためのシナリオパラメータには、ノード数、join/leave 頻度、put/get の頻度、パケット遅延 / ロスなど様々な環境を作り出すためのものが用意されている。

試験により得られる結果は、get 成功率、get 応答時間、通信帯域、の 3 つの指標である。get 成功率は、実施した get のうち既に put されている value が (すべて) 返ってきた場合の割合を算出したものである。get 応答時間は get の実行時から、すべての value が返ってくるまでの時間を平均して算出したものである。通信帯域は、ノードが送出したパケットサイズの合計をノードの join 時間の合計で割って算出したものである。

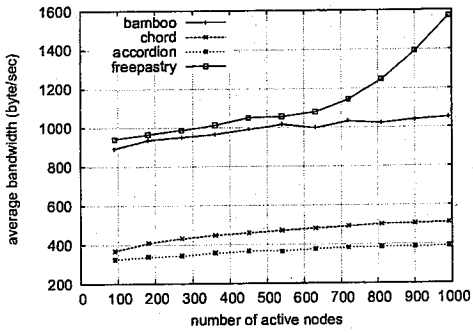

図 3: 基本実験: 通信帯域

5 性能評価実験

本節では、評価基盤を用いて既存の DHT 実装を性能評価する方法と、その実験結果に関して述べる。この実験は、最大 1000 ノード規模で、複数の DHT 実装を同一のネットワーク環境・シナリオで比較し、各 DHT 実装の性能の差や特徴を明らかにするものである。このような同一条件での性能比較は、エミュレーションによる評価基盤がなければ、従来困難であったものである。

5.1 基本構成と評価対象

評価にあたって、まず、基本となる実験構成を決めるために、「標準」シナリオを生成するためのシナリオパラメータを定めた。これは、PC の性能や数によって決まる評価基盤の性能を考慮し、評価基盤の性能限界のおよそ半分の負荷を与える程度に調整した。以降で説明する実験では、この「標準」シナリオパラメータをもとに行い、場合によってパラメータ値を変化させる。表 1に主要な「標準」シナリオパラメータを示す。

評価対象とした DHT 実装は Bamboo [8], Chord, Accrodion [6], FreePastry [2] の 4 つである。Chord と Accordion の実装は共通であり設定パラメータの違いによる。各実装のバージョンは、Bamboo: snapshot of 2006/03/03, Chord: CVS snapshot of 2006/01/27, FreePastry: versioin1.4.4 である。

これらの DHT 実装には設定パラメータがある。実装によっては非常に細かい設定ができるものもあり、それによって動作や性能に大きな違いがでる場合がある。そ

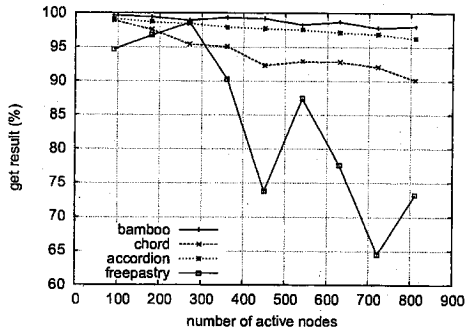

図 4: join/leave 実験: get 成功率

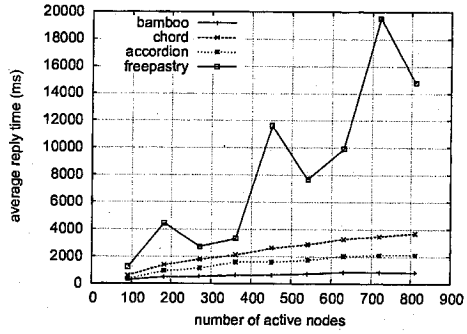

図 5: join/leave 実験: get 応答時間

こで、本評価に先立って各 DHT 実装の設定パラメータを予備実験によって調査し、高い get 成功率 (95% 以上) を維持しつつ、通信量が小さくなるものを採用した。

5.2 実験方法

- 基本実験 —最も基本的な評価として、ノードの構成に変化がない、すなわち join/leave の発生しない環境での実験を行う。DHT の特性であるスケーラビリティを確認するため、ノード数を変化させていくつかのケースで実験する。
- join/leave 実験 —ノードが参加・離脱する不安定なネットワークでの実験を行う。この実験では、ほとんどのノードが一度はネットワークから離脱するため、データが正常にノード間で引き継がれているかを確認することができる。基本実験と同様にノード数を変化させて実験する。
- データ量増加実験 —平均起動ノード数を固定にして put 回数を変化させた実験を行う。この実験により、データが増えた時の耐性を確認することができる。
- 複数 value 実験 —通常の実験では、データを *put* する際に、1 つの *key* に対し 1 つの *value* を割り当てている。本実験では、1 つの *key* に複数の *value* を割り当てる場合の実験を行う。
- 不均一実験 1 — ノードの動きに不均一性を導入するため、ノード集合を 2 つのグループに分ける。一方のグループに属するノードは put のみを行い、もう一方のグループに属するノードは get のみを行う。2 つのグループの大きさの割合を変化させて実験を行う。
- 不均一実験 2 —不均一実験 1 と同様に 2 つのグループに分ける実験だが、一方のグループに属するノードが put/get を行い、もう一方のグループに属するノードは put/get を行わない。この場合でも、データそのものは全ノードに及ぶことになる。

各実験は 1 回のみ行い結果をグラフにプロットした。

5.3 実験結果

- 基本実験 — get 成功率 (図 1) はすべての DHT で高成功率 (95% 以上) を維持している。get 成功率が 100% にならない理由は、ノードが参加してからそのノードのルーティングテーブルが完全に構成されるまでの間で put や get が失敗しているためである。ネットワークのサイズが大きくなるとルーティングテーブルが完全に構成されるまでに時間がかかり、より多くの put や get が失敗するため、get 成功率のグラフは右下がりになる。FreePastry は、多くのケースでほぼ 100% となっているが、これは、キャッシュ等の機能がうまく働いているものと推測される。get 応答時間 (図 2) は、どの DHT でも似た上昇傾向を示しているが、Bamboo, Accordion, Chord, FreePasty の順に応答性がよい。FreePastry はノード数 (横軸) が 900 を越えると、応答性が急に悪くなり、図 1 と照らし合わせると、ここに何からの実装上の限界があると推測できる。最後に、通信帯域 (図 3) は、Chord と Accordion が低い値を維持しているのに対して、FreePastry についてはノード数が大きくなると、傾きが大きくなり他の DHT の傾向と乖離している。
- join/leave 実験 —基本実験とは異なり、get 成功率 (図 4) は全体的に低下している。特に、FreePastry は極端に低下し、また結果のばらつきも大きくなっている。ここから類推できることは、FreePastry にとって今回のシナリオは頻繁すぎる join/leave だという

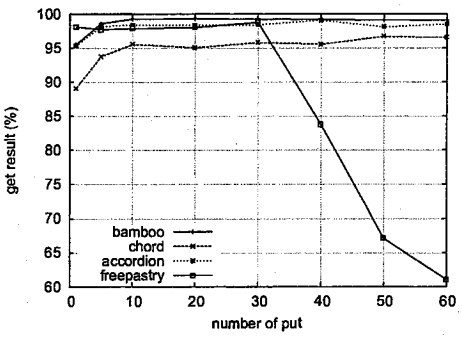

図 6: データ量増加実験: get 成功率

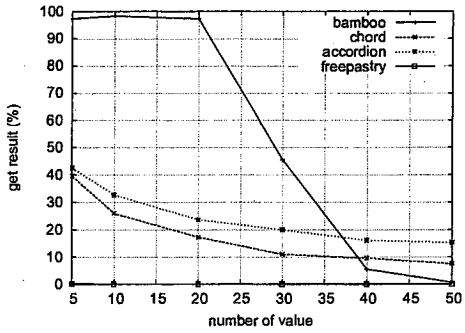

図 7: 複数 value 実験: get 成功率

ことである。 get 応答時間 (図 5) をみても、平均起動ノード数 (横軸) が 400 を超えたあたりから大きく上昇し、基本実験のときの性能がでていないことが分かる。

- データ量増加実験 —図 6の結果は、横軸が 1 ノードあたりの put 回数、縦軸が get 成功率を示している。 FreePastry は put 回数が 30 を越えたところから get 成功率の低下がみられる。 Bamboo, Chord, Accordion は正常に動作している。
- 複数 value 実験 —図 7の結果は、横軸が 1 つの *key* あたりの *value* の数で、縦軸が get 成功率を示している。 FreePastry は常に get 成功率が 0% で複数 value をサポートしていないことが分かる。 Chord と Accordion は get 成功率が十分に得られていないが、 0% になることはない。 Bamboo は 20 value までは正常に動作するが、それより大きくなると極端に悪化し、 50 value ではほぼ 0% となる。これから、 Bamboo は複数 value の get に関して、何からの実装上の制約があることが分かる。
- 不均一実験 1 —図 8の結果は、横軸が put を行うノードのグループの大きさの割合、縦軸が通信帯域を示している。 Bamboo の通信帯域はほぼ一定であるが、 Chord, Accordion, FreePastry の通信帯域には減少傾向がある。これは put にかかる通信量が get にかかる通信量より十分に小さいためと考えられる。
- 不均一実験 2 —図 9の結果は、横軸が put/get を行うノードのグループの大きさの割合、縦軸が get 成功率を示している。すべての DHT において高い成功率 (90% 以上) を維持しているが、その中でも FreePastry の安定性がよい。ネットワーク全体で put/get を行うノードが少ない場合は、 FreePastry は有利であることが分かる。

5.4 考察

以上の実験から得られた DHT 実装の特性を述べる。

Bamboo はどの実験においても安定的な結果となり、特に応答時間が短いことが特徴である。しかし、通信帯域は大きい部類に属し、低効率となっている。よって、通信帯域に制限がなく、安定性・応答性を重視するアプリケーションでの利用に適している。

Chord と Accordion は、 Bamboo より大幅に小さい通信帯域でその効率性が特徴となっている。しかしながら、 Chord の応答時間は Bamboo よりも長く、 [5] でも報告されているように、応答時間と通信帯域にトレードオフの関係があることを示している。 Accordion はすべての実験において Chord の性能を上回るため、現段階で、 Accordion の代わりに Chord を使う理由は見つけられない。 Accordion は、通信帯域を抑えたい場合で、成功率の低下が致命的とならないアプリケーションでの利用に適している。

FreePastry は、 join/leave のない環境や put/get があまり起こらない環境、つまり、静的に近いネットワークにおいて性能がよい。よって、基本的に静的ネットワークを想定したアプリケーションでの利用に適している。

6 エミュレーション評価の有効性

5節の実験では、エミュレーション評価を行うことで、基本実験の FreePastry や複数 value 実験の Bamboo のように、実装上の性能の問題点を発見することができた。以下では、性能の問題以外でエミュレーションによって明らかになった 2 つの点について説明する。

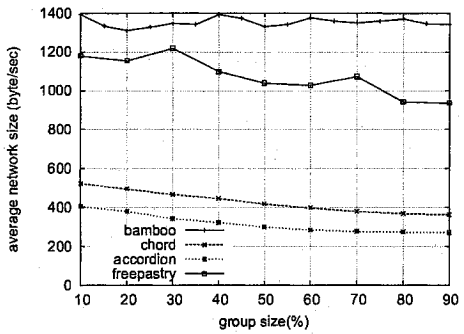

図 8: 不均一実験 1: 通信帯域

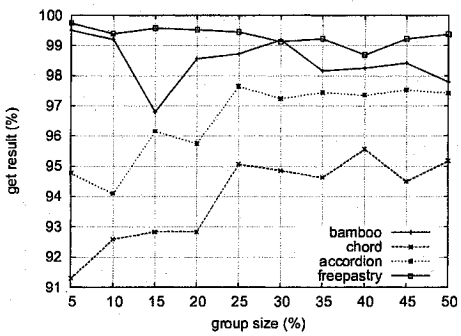

図 9: 不均一実験 2: get 成功率

- Bamboo と FreePastry はいずれも同じ DHT アルゴリズム Pastry をベースに実装されている。ところが今回の実験結果によると、ノードの参加・離脱が起こらない基本実験では、両者はほぼ変わらない性能を示すが、ノードの参加・離脱が起こる他の実験では大きく差がでて、Bamboo が優位な結果となっている。これは、Bamboo の設計コンセプトがノードの参加・離脱を定常イベントとして扱うことによるものと考えられる。
- 基本実験での Chord と Bamboo の応答時間に着目すると、多くのケースで Bamboo の方が短く、ほぼ 2 倍の開きがある。これは、Bamboo が再帰問い合わせという方式を採用しているのに対して、Chord は逐次問い合わせという方式を採用しているためと考えられる。

これらの差は、アルゴリズムを評価するのみでは発見できず、実装レベルの差となっている。この結果から、実装の特徴を含めた DHT の評価にはシミュレーションでは不十分で、エミュレーションが必要であることが分かる。

7 おわりに

本稿では、DHT の実装を評価するにあたってエミュレーションによる評価の必要性を主張し、エミュレーションを使って複数の種類の実験を行った。その結果、アルゴリズムのみの評価では分からない差をみることができた。DHT 実装の性能評価は、特にアプリケーションを作る場合は重要で、アプリケーションの要件を満たす DHT 実装を探したり、DHT の性能に合わせてアプリケーションを設計したりするのに有用であると考えている。エミュレーション評価により、今後の DHT の研究開発やアプリケーション開発が促進されることを期待する。

参考文献

[1] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[2] http://freepastry.org/.

[3] http://pdos.csail.mit.edu/p2psim/kingdata/.

[4] http://www.bittorrent.org/.

[5] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, February 2004.

[6] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth efficient management of DHT routing tables. In *the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[7] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS02, Cambridge, USA*, March 2002.

[8] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[9] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat. MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks",. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware 2001*, 2001.

[11] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pp. 149–160, 2001.

[12] 加藤大志, 神谷俊之. DHT 性能評価法の提案と評価基盤の構築. 情報処理学会研究報告, 2006-DSM-40, pp. 31–36, 2006.