SkipNet: A Scalable Network with Practical Locality Properties

Nicholas J.A. Harvey, John Dunagan, Michael B. Jones,
Stefan Saroiu, Marvin Theimer, Alec Wolman

2004/11/04 IDEON camp memo

1 概要

Skip List を利用した DHT。特徴は ID ベースの検索のみならず名前ベースの検索が可能な事と、あるドメイン (名前空間) 内部でのメッセージの配送 (content locality) や負荷分散 (CLB: Constraind Load Balancing) の設定が可能な事。

1.1 原理

Aspnes の論文と同様、Skip List を利用している。Skip List 自身は、Perfect Skip List (あまり実用的ではない) と Probablistic Skip List (実用的) とがある。Probablistic Skip List では、それぞれの node が確率的に高さ h を持つ。高さ h となる確率は、$\frac{1}{2^h}$ である。

SkipNet の度数は N をノード数とした時に $2logN$ である。ノードには恣意的な (DNS name のような) 名前: name ID をつける。名前空間を Ring とした時に、R-Table(Routing Table) は各階層 h において 2^h 右ないし左のノードへのポインタを含む。階層 $h+1$ のネットワークは、階層 h のネットワークをランダムかつ一様に分割する事で得られる。

numeric ID とは、node、かつ階層 $h > 0$ のネットワークに与えられた ID である。階層 h のネットワークは bit 長 h の numeric ID を持つ。全ての階層に所属するネットワークは、root level (全ての node が所属する) を根とする二分木状に分布し、例えば、ネットワーク 010 は $h = 3$ で、0 を左とすると、左→右→左と二分木を下った場所にあるネットワークとなる。

また、node は自分の名前の digest (実際は均一な乱数なら何でもいい) を自分の ID として使い、その ID の最初の h bit の ring に所属する。階層 h のメンバが 1 つ (両側のポインタが自分を指している状態) で R-Table の作成は止まる。

Chord との対比: Chord では、h 番目の finger は、numeric ID 空間において 2^h 番目のノードを指す (yd: あるいは、それに相当する – h の数えかたがちょっと難しい)。SkipNet では、h 番目のネットワークは名前空間で 2^h 先のノードを指し、同時に $h-n$ 番目のネットワークでは、numeric ID 空間で 2^n 先のノードを指す。

Name ID による経路づけ: 各ノードで、目標を通りすぎない最も高位のポインタを利用する。root ring から名前順で sort されているので、あるサブドメインの内部で経路づけした場合、その名前空間の中のノードのみを通る事が保証される (yd: underlay で外通ってるかどうかは関知されない)。

Numeric ID による経路づけ: ほぼ Plaxton Mesh。まず、root ring を順に巡り、最初の 1 bit が一致するノードまで行く。次に階層を上

がり ($h = h + 1$)、$h + 1$bit が一致するノードまで行く。以下繰り返し (一致するノードが見つからなければ最適マッチなノードで終了)。

(yd: データはどこに置くんだろう……。name / numeric のどっち？)

Node Join: Node の numeric ID による経路付けで、Node が入るべき Ring を発見する。Join した最上位 (h) の Ring の neighbor は、Name ID 順における 2^h むこうの neighbor になる。ここで、階層を下りながら、それぞれの階層での Name ID 順の neighbor を発見していく。最後に、root Ring から順番に Join して行き、h 階層目まで join したら終了。Pseudocode では、最初の Insertion メッセージを飛ばす時に各階層の neighbor を探している模様。

(yd: Insert 途中の検索とか、データ責任範囲の受け渡しはどうなるのか)

Leaf set: successor list と一緒。左右に $2/L$ ノード確保している。L は定数 (= 16)。

(yd: 他の研究を見ていると、$L = O(logN)$ とするのが良さげ)

Background Repair: $h = 0$ からはじめて、$h+1$ の左右のノードまで traverse する方法と、それぞれの階層において left/right pointer を交換する方法の二種類。Root Ring さえきちんと管理されていれば、多少狂っていても効率が落ちるのみで問題無し。

Network partition: 名前で root ring をソートしているので、partition には強い。つまり、ばらばらになるのではなく、root ring においてざっくり割れる。結合する時もきちんと結合できる。(FIXME: Section 6 の内容を足す)

(以下 section 5 以降)

Dense Table: numeric ID の数え方を binary じゃなくて k-ary にすれば、state 数は減りますが、hop 数は増える。一方、k-1 個の前後 node を各階層で持つ (dense R-Table) ようにすれば、逆の効果が得られる。

Duplicate Pointer Elimination: h と $h+1$ の Pointer が同じ場合は、$h+1$ の Pointer はより遠くを指すように変更しても良い。単純な改良だけど、それで 25%の性能向上が得られる (yd: 何の性能かわからないけど、恐らく経路数か、後述する RDP)

Virtual Node: FIXME

1.2 性質

Security: name boundery を押えられる、というリスクがある。`microsoft.com` に対して `microsofa.com` を取れれば、左から入ってくるトラフィックのある程度が押えられるかもしれない。

: range queries: 名前順に sort されているので、名前付け規則をきちんとやればうまくいくと考えられる。

以下、順序はちょっと前後しますが特出しな話題。

1.3 Locality

Proximity でない所がポイント。つまり、SkipNet は (Name ID の正しさが検証されている限り) 名前のある部分木の中に探索や負荷分散を収める事ができる。

CLB(Constraind Load Balancing) では、データの ID をドメイン部 (CLB domain) と suffix(CLB suffix) に分割した上で、CLB domain に含まれるノード内で CLB suffix の digest を用いて (search by node ID) データを配置する。検索も同様で、結果的に name ID search + node ID search が必要になる。node ID search は、名前空間の境界線で折り返す必要が出てくる場合があるので、最大で通常の単方向 node ID search の二倍のホップ数が必要になる。

1.4 Network Proximity

(Section 5.3 から) **P-Table**: Network Proximity を計測するテーブル。階層 h の P-Table は、$2^h < x < 2^{h+1}$ 番目のノードのうち、ネットワーク距離的に近いノードを持つ。

作りかた:

1. R-Table の内容をコピーする。
2. j node を含む、P-Table Join Message を作り、seeding node に送る (yd: j とは何?) Join Message には j-1 の interval(間隙?) がある
3. P-Table Join Message を受信した node は、自分の P-Table から Join Message の interval に入るべき node を選択し、入れる。もし interval に空きがあったら、適当な node(?) に送る。(yd: joining node から最も遠い node に送ると言っているが、ここの far は name ID なのかそれとも latency なのか?)
4. あとは適宜 update していく

C-Table: 距離付き Plaxton Mesh のようなものか?

1.5 分析

ごめんなさい、延々と証明が続くだけなのでとりあえず SKIP

1.6 実験結果

比較対象: Chord, Pastry, "basic" SkipNet with R-Table, "Full" SkipNet with C-Table and P-Table (sparse and dense R-Table).

トポロジ: Mercator[36] / GT-ITM[39]

測ったもの: RDP (IP 上の latency と、overlay 上の latency の比 incl. ノード上での処理時間), Physical network distance(IP 上の hop 数), Number of failed lookups(failure 時の問い合わせ失敗)

組織モデル: unform size distribution, Zipf distribution

node locality: unform, cluster (within AS or stub network), zipf (single center and Zipf relative distance)

host/organization: host name → MS 社内より収集, organizations → Gnutella 参加組織から収集

Fig. 12 を見ると、Pastry はやはり優秀。routing entry の数 (Tbl. 1) を見ると Chord はやはり少ない。

Fig. 13 は local な query が増えるほど問い合わせの hop 数が減る様子を示している。hop 数だけでは transit link のコストが表現できないが、GT-ITM ベースのシミュレーションでは Pastry の 1/7(100% local の場合) と主張している。

Fig. 14 は、全体の 15%を構成するネットワークが分割された場合の動作を示している。通常の DHT は 85%のノードが死んだら問い合わせできないが、SkipNet は local な問い合わせならば Ok だという事を示している。

Fig. 15 が何を計っているのかちょっと不明。複数の AS が結合された後の hop 数らしいが、どことどこの hop 数か、overlay か underlay かが書かれていない。

Fig. 16 は CLB の様子。C-Table は効いてるが、Pastry に比べてまだ劣っている様子が見て取れる。

Fig. 17 は P-Table の評価。同じ条件で k を変更してどのように RDP が変化したかを検証している。

2　コメント

かなりイケてる。しかし、MS というぐらいでぱてられてる可能性大？state が多かったりテーブルの扱いが複雑だったりするという問題はあるが、テーブルの実装が異なるノードが混在した場合 (R-Table は共通)、何が起きるかにやや興味有り。

Chord の Proximity 話もいくつかあると思うが (incl. narupy) それらとの比較が全く語られていないのがちょっと Chord もかわいそうな気がしている。