2004年 7月 27日(火)

Kademlia

首藤一幸
産業技術総合研究所 グリッド研究センター

資料

- IPTPS02 の論文とスライド
  - "Kademlia: A Peer-topeer Information System Based on the XOR Metric"
- ウェブページ
  - http://kademlia.scs.cs.nyu.edu/
    7/26時点では参照できず。

Kademlia の位置付け

- 論文中に現れる比較対象:
  - Chord
    - Chord の悪い点:
      - 経路表が厳格。
        - ノードの failure や、経路表の欠損からの復旧が複雑。
      - qeury 転送の方向が一方向。
        - Incoming traffic を元に経路表を更新できない。
    - Kademlia では、2ノード間の関係は symmetric。
      - x が y の経路表に載ってれば、y は x の経路表に載ってる。
      - incoming traffic を元にして経路表を更新できる。
  - Pastry
    - Pastry の routing は、途中で、Plaxtonの方法から leaf set を使った方法に切り替わる。
    - Kademlia は、最後までひとつの方法を使い通す。
- Kademlia の特徴
  - ノードが頻繁に出入りする状況を想定している。
    - ランダムに選んだノードが 1時間後にも online である確率は 1/2。

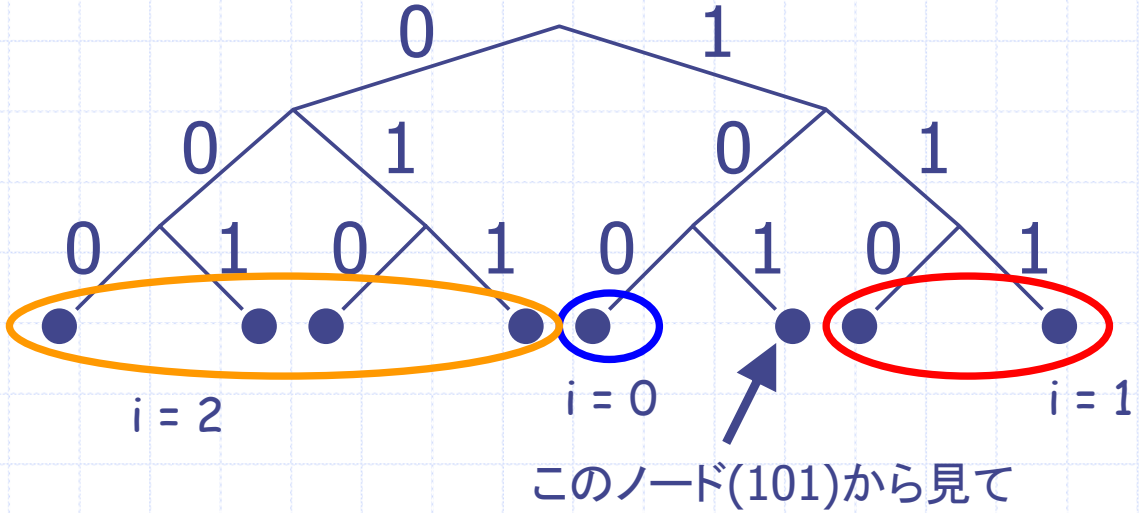
Key とノードID

- Key, ノード ID は 160 bit。
  - やはり SHA-1 の使用を想定している。
  - 「ノード ID はChord と同様に決める」
    - 論文では、話を簡単にするために、単にランダム。
- ノード間の距離として、XOR を採用。
  - $d(x, y) = x \oplus y$
  - 例: 11111 と 11010 の距離は 101 (十進では5)
- XOR → symmetric
  - 経路表中のノードから、逆に query を受け取ることになる。
  - request, reply、あらゆる種類のメッセージ受信を契機に、経路表を更新できる。する。
    - → 状態管理用のメッセージは不要。
    - ただし、query が少ない場合には、経路表中の参照されない部分が更新されないので、ノードが自発的に query を出す (reflesh)。
  - cf. Chord

経路表

- 160 本の k-buckets
  - k-bucket
    - 自身との距離が [2^i, 2^(i+1)) (0 <= i < 160)であるノードのコンタクト先リスト。
      - コンタクト先: <IPアドレス, UDPポート, ノードID>
    - リストの長さは最長 k。論文では k = 20。
    - 参照された時刻が古い順にソートされている。
      - 参照されたエントリは、末尾に移す。

Key, ノードID が 3 bit だとする:



i	ノード101 からの距離	k-bucket の 内容例
0	1	100
1	2 ～ 3	110, 111
2	4 ～ 7	001, 011

AIST

経路表の更新

- 何かしらのメッセージ (request/reply) を受け取った時点で更新を行う。
  - 送信元が k-bucket に入っている場合、そのノードを k-bucket の末尾に移す。
  - 送信元が k-bucket 中に存在しない場合:
    - 当該 k-bucket 中の先頭のノードが online か否かを確認する。
      - PING（後述）で確認。
    - online なら、先頭のノードを残す。
      - つまり、古い方を優先する！
      - なぜなら、これまで online だった古いノードの方が、今後も online である確率が高いから。
        - Gnutella の trace データを解析した結果。

プロトコル

- query は転送するのではなく、すべて、問い合わせ元から RPC (remote procedure call, 遠隔問い合わせ) する。
  - 非同期 RPC
    - 返答を待たない。
- 問い合わせは 4種類:
  - PING
  - STORE
  - FIND_NODE
  - FIND_VALUE

プロトコル

- 4種類の問い合わせ
  - PING
    - online かどうか問い合わせる。
  - STORE (key, value)
    - 対象ノードに <key, value> ペアを保持させる。
  - FIND_NODE (key)
    - key に (XOR 距離が) 近いノード k個のコンタクト先を問い合わせる。
    - 要求を受けたノードは、該当する k-bucket の中身を返す。
      - 当該 k-bucket が満杯でない場合、周辺の k-bucket からも近接ノードを選ぶ。
  - FIND_VALUE (key)
    - FIND_NODE と同じ。
    - ただし、問い合わせ先ノードが key に対応する value を保持している場合は、コンタクト先ではなくて value が返される。

発見

- 与えられたノード ID に近いノード k個 (のコンタクト先) を得る。
  - FIND_NODE を繰り返して、長さ k のリストを refine していく。
    - k は、ターゲットから近い順に並べておく。
    - ターゲットから近い α 個（後述）のノードに対して問い合わせる。
      - ただし、問い合わせ済みのノードは除く。
    - ノード k 個すべてに対して問い合わせ済み、という状態になったら終了。
  - 非同期 並行 問い合わせ
    - 同時に α 個のノードに問い合わせを行う。
      - すべての返答を待たず、その時点で得られている返答を使って処理を進める。
      - 論文には、α は「例えば 3」とある。
      - ノード k 個中から α 個を選ぶ方法は、この論文の範囲外。
        - 例えば、ネットワーク的に近いノードを選ぶ。
          そのために、k-bucket 中の各エントリに RTT を付けておく。
- key に対する value を発見したい場合は、FIND_NODE ではなく FIND_VALUE を使う。

ノードの新規参加・離脱

- 新規参加
  - 既存ノードの経路表を借りる。
  - 自分自身のノード ID を lookup する。
    - FIND_NODE を受けた側は、自分よりも問い合わせ元の方が持つべきインデックス情報を、問い合わせ元に対して STORE する。
- 離脱
  - 特に何もする必要なし。

インデックス情報の管理

- インデックス情報: <key, value>ペア
- Publish
  - key に最も近いノード k個を得る。
  - それらに対してインデックス情報を STORE する。
- 生存管理
  - ノードは、自身が持つインデックス情報を 1時間に 1度 publish する。
    - 新規参加ノードにも持たせることで、発見され易くする。
    - インデックス情報の消失を防ぐ。
  - Original publisher は、24時間に 1度、再 publish する。
    - Original publisher による STORE を受けなかった場合、インデックス情報は expire する。
  - さらなる最適化も考えられるが、それは論文の範囲外。