Koorde: A simple degree-optimal distributed hash table

M. Frans Kashoek and David R. Karger
MIT Laboratory for Computer Science

2004/11/04 IDEON camp memo

1 概要

1.1 位置付け

de Bruijn graph を用いた DHT の実装。論文の構成は、DHT 性能の下限を求めた上で方式を語る形式。

1.2 DHT の下限の議論

度数 (d) とホップ数の関係: 最低でも、$log_d n - 1$(最長経路)、$log_d n - O(1)$(平均) ホップ必要。[1]

ここから、度数が固定の場合は $O(logn)$ ホップ、度数を $d = O(logn))$ とする事で、$O(log_{O(logN)} logN) = O((logn)/loglogn)$ ホップである。

接続性とノード故障との関係: 全てのノードが 1/2 の確率で故障する時、一部のノードは $\Omega(logn)$ の度数を持つ必要あり。[2]集中を避けるためには、全てのノードが $\Omega(logn)$ の度数を持つ事が望ましい。

[1]正直、証明は理解してない

[2]これも証明は理解していない。neighbor of neighbor に isolated node が含まれる可能性を計算しているように見えるが……

> $\Omega(logn)$って何？ (from WikiPedia):
>
> $$lim_{x\to\infty}\frac{f(x)}{g(x)} = 0$$
>
> の時、$f(x) = o(g(x))$。で、Ω はその否定。端的に言えば、どんなに絞っても g(x) は f(x) を下回る事はない、という事か。

その後、maintainance traffic について同様の議論を展開している。“half-life” の論文 [7] を参照。また、負荷分散についてはトレードオフで、$O(logn)$ に virtual node を作る事によって良好な負荷分散を得られるのだが、一方でノードのステートが増えてしまい、理想的な hop では無くなるという問題がある。

1.3 Koorde の方法

Chord の上に de Bruijn routing を実装した。その結果、各ノードのステート量は $O(1)$ にできる。記法や ID 空間の表現は Chord と一緒。

ノードの ID を m< 2^b とすると、各ノードが持つ edge は m¡¡1)+i)&((1¡¡b)-1)*(ただし i は 0 か 1) へ向かう。というか、bit 列を左に 1 shift して、最上位桁は捨てて、最下位桁に i(0 か 1) を埋める。

実際には、2^b の ID 空間全てにノードが来る筈もないので、仮想ノード (imagenary node) を置く。i は de Brujin path を辿る。しかし、de Brujin path の predecessor にメッセージは転送される。

procedure の出口は 3 通り。successor を返すか、d(predecessor of 2m) に飛ぶ (de Bruijn path) か、successor に飛ぶ (args かわらず) か。ただし、lookup のアルゴリズム (Figure. 3)i が m と $successor(m)$ の間に来る状態でないと de Brujin path は通らない。 → ？

de Bruijn を一回、successor を二回通る (期待値) らしい。

2 コメント

正直、`imagenary node` が意味不明。本当に動くの？

3 メモ

サンプル書こうとしたけど、imagenary node の選択がわからない

各ノード m は、de Bruijn node に加えて、$successor(m)$ と、$d : predecessor(2m)$ を持つ。例えば、$b = 4$ の時、ノードが 0000, 0010, 0100, 0110, 1000, 1010, 1100, 1110 の 8 つだったとする。ここで、$k = 0001$ を $m = 1100$ が発見するとする。

- `1100.lookup(k=0001, kshift=0001, i=1100)`
  - `k` は 1100 から 1110 の間に入らない
  - `i` は 1100 から 1110 の間に入らない
  - ↓
- `1110.lookup(k=0001, kshift=0001, i=1100)`
  - `k` は 1110 から 0000 の間に入らない
  - `i` は 1110 から 0000 の間に入らない
  - ↓
- `0000.lookup(k=0001, kshift=0001, i=1100)`
  - `k` は 0000 から 0010 の間に入る
  - return(successor)

初期値の i か、k かのどちらかが successor と m との間に来るまで、ひたすら successor を辿るように見えるのだが……。

ちなみに de Bruijn のつもりで巡るものだとすると、(i,k,kshift) → (1100,0001,0001) → (1000,0001,0010) → (0000,0001,0100) → (0000,0001,1000) → (0001,0001,0001) と巡るはず。なので、i が 1100 と 1110 の間に含まれる可能性は無い。1100 の d は 1000 なのだけど。

4 ページ目の説明を自然に考えれば、(i,k,kshift,m) → (1100,1010,1010,1100) → (1001,1010,0101,1000) → (?) → (0010,1010,1010,0010) → (0101,1010,0101,0100) → (?) → (1010,1010,1010,1010) となるように経路を組む、という事だと思う。

imagenary node が m と successor との間にある場合のみ de bruijn に飛ぶ、という事はどういう事なのか？