Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems
Antony Rowstron, Peter Druschel

1. Introduction

2. Design of Pastry

ノードには 128bit の nodeId
(128-bit node id is assigned to each node)
  ランダムに割り当て
  (randomly)
  たとえば IPアドレスのハッシュ
  (e.g. hash of IP address)
0 ~ (2^128)-1 の環に均一に分布
(uniformly distributed around a ring)

隣接する nodeId を持つノードは, 近くではない
(adjacent nodeIds are diverse in real IP networks)

検索対象のキー も 128bit
(keys are also expressed in 128 bit)

そのキーに(数値的に)最も近い nodeId を持つノードを探すのが目的
(route query to the numerically closest node to a given key)

128bit の二進数を, b bitごとに区切る (b ~ 4)
(nodeIds and keys as a sequence of digits with base 2^b)
  2^b 進数 としては, 128/b 桁
  ex. 16進数としては 32桁
  (e.g. if b=4, 128 bit is expressed by 32 hexadecimal digits)

ノード総数 N (number of nodes is N)
探索のホップ数は log_{2^b}(N) 以下
(necessary steps are under log_{2^b}(N))
  log_{2^b}(N) = (1/b)log_{2}(N)

routing step の概要
  現在ノードの nodeId と key の共通 prefix の長さ L
  (先頭から何桁一致しているか)
  (L: number of digits of common prefix between the nodeId of the local node
   and the key)

  L+1桁(以上)一致しているノードに転送する
  (route the query to the node whose nodeId has at least L+1 prefix with the key)

  転送のたびに一致桁数が増える
  桁数のオーダで目的地に到達
  (will arrive at the destination in the order of the number of digits)

2.1 Pastry node state
みっつの表 (three tables)
  routing table
  neighborhood set
  leaf set

routing table, R
  log_{2^b}(N) 行(rows) （桁数ぶん 128/b）
  各行は 2^b-1 エントリ(entry) （一桁の数値ぶん）
    (値:ノードへの参照 -- つまり nodeId, IP addr)

  第n行の各エントリの指すノードのnodeIdは (nodeId in an entry in the row n)
    自ノードと先頭n桁が等しく (has n common prefix with the local node)
    n+1桁は異なる (digit n+1 differs)
    n+1桁めが何であるかで, 2^b-1とおり
    (2^b-1 entries according to the digit n+1)

    自ノードと実際に近いもの(proximity metricによる)を選ぶ

    (entry is chosen according to the real proximity)

  該当ノードがない場所は empty
  (entry may be empty)

  平均的にノードが分布しているとすると
  (if nodes are distributed uniformly,)
    log_{2^b}(N) 行だけが埋まる
    (rows up to log_{2^b}(N) will be filled)
    (共通桁数があまり多い所は埋まらない)

  bの決定はトレードオフ (choice of b involves a trade-off)
    routing table の(埋まった)エントリ数 (populated portion of the routing table)
      log_{2^b}(N) * (2^b-1)
    最大ホップ数 (maximum number of hops)
      log_{2^b}(N) = (1/b)log_{2}(N)
  例: b=4
  ノード数 10^6 -> エントリ数75, ホップ数5
  ノード数 10^9 -> エントリ数105, ホップ数7

Fig. 1 (b=2, nodeId 16bit -- 四進8桁)

NodeId 10233102

Leaf set   small|large
10233033 10233021 10233120 10233122
10233001 10233000 10233230 10233232

Routing table
-0-2212102 1   -2-2301203 -3-1203203
0   1-1-301233 1-2-230203 1-3-021022
10-0-31203 10-1-32102 2   10-3-23302
102-0-0230 102-1-1302 102-2-2302 3
1023-0-322 1023-1-000 1023-2-121 3
10233-0-01 1   10233-2-32
0   102331-2-0
                      2
Neighborhood set
13021022 10200230 11301233 31301233
02212102 22301203 31203203 33213321
----
routing table
 (共通部分n桁 - n+1桁め - 残り)と表示してある
 (n common prefix - digit n+1 - rest)
ひと桁の数の場所は, 自ノードの場所
(one digit indicates the local node)

neighborhood set M
  自ノードと proximity metric 的に近いノード
  (closest nodes according the proximity metric)
  routing には普段は使わない
  (not normally used in routing)

leaf set L
  nodeId が近いノード
  (numerically closest nodes)
  自ノードから大小同数ずつ (larger and smaller, with the same number)

L, M のサイズはたとえば 2^b or 2*2^b

2.2 Routing
Table 1. Pseudo code
R^{i}_{l}: routing table の l行 i番目のエントリ (row l, column i)
L_{i}: leaf set の i番目に近いノードの nodeId (i-th closest nodeId)
  i < 0 なら小さい側 (smaller)
  i > 0 なら大きい側 (larger)
D_{l}: キー D の l 桁目 (l-th digit in the key)
shl(A,B): 二つの Id の先頭からの共通桁数

(length of the prefix shared among A, B)

コードの説明 (the code)
(L.1) キーが leaf set の範囲内に収まるか?
(within range of the leaf set?)
そうなら, leaf set から最も近いノードを選び, 転送して終了 (L.3)
(forward to L_{i})

(L.6-8)
leaf set の範囲外なら, routing table を使って,
(out of the range of the leaf set -- use routing table)
現在より共通桁数の多いノードに転送する
(forward to the node with more common prefix digits)

(L.11-14)
routing table の該当位置が empty のとき
(if the entry in the routing table is empty)
キーとそこまでidが一致するノードがない
(there is no node whose id matches the key at that digit)
あるいは該当ノードが unreachable のとき
(or the node in the entry is unreachable)
すべての表に含まれるノードのうち
(out of all known nodes in the three tables, forward to the node which)
共通桁数が現在以上 (has same or longer common prefix)
nodeId が現在より近い (and is closer to the key)
ものをえらび転送

最後の条件を満たすノードは, leaf set のどちらか側に存在
(in the last case, the node is in the leaf set)
どちらか側が全滅しない限り routing 成功
(routing will succeed unless all the half of the leaf set is unreachable)

この routing procedure は収束する
(this routing procedure always converges)

Routing Performance
routing table が正確 (accurate)
故障ノードなし (no node failure)
の条件の元で
必要なステップ数は log_{2^b}(N)
(expected number of routing steps is log_{2^b}(N))

explanation:
三つの場合分け (three cases)
○第一(use leaf set)
1ホップで完了 (at most one hop)
○第二(use routing table)
ステップ毎にキーの共通部分がひと桁ずつ増加
(length of the prefix increased in each step)
○第三
leaf set になし, 指定桁数の共通部分を持つノードがいない
(not covered by the leaf set, and there is no routing table entry)
|L|を適当に取ればその確率は低いし (rare)
1ステップよりも多く必要な場合はほとんどない
(no more than one additional step with high probability)

多数のノードが同時に故障すると, 最悪 N ステップになる
(with many simultaneous node failures, it takes at worst N steps)
実際には, 故障ノード数にしたがって次第に悪化 (実験(3.1)によれば)
(in practice, the performance degrades gradually with the number of failures)
連続する |L|/2 ノードが故障しなければ, 届く
(delivery is guaranteed unless |L|/2 consecutive nodes fail)
|L|を適切にとれば, その確率は低い.
(the probability can be made very low)

2.3 Pastry API
省略

2.4 Self-organization and adaptation

新規ノードの参加と離脱
(arrival and departure of nodes)

参加 (node arrival)
やることは
  自分の表(みっつ)の設定 (initialize three state tables)
  他ノードへの通知 (inform other nodes)

初期接続 (initial connection)
  proximity metric にしたがって, 近いノードに接続
  (we assume the new node knows initially about a nearby Pastry node)
    "expanding ring" IP multicast
    管理者からの情報 (by the system administrator)
新規ノードの nodeId を X とする (自分で計算)
 (the new node X computes its nodeId)
 hash of IP address OR host key
初期接続先 A (initial node A)

X は A に join メッセージの転送を依頼
(X asks A to route a join message with the key equal to X)
  キーとして, X を含む
  X に nodeId が一番近いノード Z が見つかるはず
  (the message will be routed to the existing node Z, numerically closest to X)

join を受信したノード(A, ..., Z)は自分の表を X に送信
(A, ..., Z) will send their state tables to X)

X はこの表を元に自分の表を作る (すぐ後で説明)
(X initializes its state tables, as described below)

最後に, X は関係するノードに参加を通知
(finally, X informs nodes of its arrival)

さて, 表の構成法 (state table initialization)
neighborhood set
  A は X に隣接しているから
  (becase A is in proximity to X,)
  A の neighbor set をちょっと修正すると X の neighbor set になる
    A 自身を加えて, ひとつはずす
  (use A's neighbor set to initialize X's)

leaf set
  Z は X の nodeId に最も近いから
  (becase Z has the numerically closest nodeId to X,)
  Z の leaf set をちょっと修正すると X の leaf set になる
    Z 自身を加えて, ひとつはずす
  (use Z's leaf set to initialize X's)

routing table
まず第0行から - 共通桁数 0 (row 0 - zero common prefix)
  じつは第0行はノードに依存しない (row 0 is independent of nodeId)
  A の第0行がそのまま使える (A_0 can be used as X_0)
    proximity metric としても有利
  空欄があるなら A 自身をそこに埋めればよい

第1行 (row 1)
  AとXの共通桁数が 0 だとすると (if A and X have no common prefix)
  A の第1行は役に立たない (A_1 is of no use for X_1)
    先頭桁が違うから
  ここで, join メッセージをAから受けたノード B に着目
    Bの先頭桁が X と一致したからこそ B に転送した
  (node B has at least one common prefix with X)
  B の第1行を使えばよい
  (B_1 can be used as X_1)

以下, join メッセージの中継ノードの情報から, 順に第i行が使える

(and X_2 from C_2, ...)

他ノードへの通知 (inform other nodes)
  X の三つの表に登場するノードへ, 表を送信
  (X transmits a copy of state tables to the nodes appeared in three tables)
  それを受けとったら適切に更新

参加で必要なメッセージ数は O(log_{2^b}(N))
  定数は 3 * 2^b

concurrent node arrival
  そもそも競合は起こりにくい
    参加(離脱)で影響を受けるノードが少ないから
  (since arrival/departure affects only a small number of nodes,
   contention is rare)

  ノードAがノードBに情報を送る際にタイムスタンプを付加
  (A attaches timestamps to messagess to B)
  Bが自分の情報を更新して, Aに通知を送る際に
    そのタイムスタンプを添付
    (B attaches the original timestamp when it sends update message to A)
  Aは自分の情報に変更があったかを確認
  (A can check if its state has since changed)
    変更があったら再びBへ知らせる
    (if changed, send an update to B again)

ノードの故障, 離脱 (node departure)

  予告なし (may fail without warning)
  nodeId で隣接するノードからアクセスできなくなった時
  (when immediate neighbors in the nodeId space can no longer communicate
   with the node)

故障ノードの置き換え (replace a failed node)
leaf set
  故障ノードの側で, 最も離れたノードからleaf tableをもらう
  (asks the most distant node in the set for its leaf table)
  この中から適当なノードを自分のleaf setに補充する
  (chose the appropriate node to insert into L)
    本当に生きているか確認してから
    (verifying that the node is actually alive)
  これがうまくいかないのは, 片側が全滅した場合のみ
  (this fails only when all |L|/2 have failed simultaneously)
  nodeId空間はばらばらなので,おきにくいはず
  (very unlikely, because the diversity of nodes)

routing table
  そこにルーティングしようとした時に発覚
  (detected when that node attempts to contact the failed node)

  その場合, ルーティングは別ノードに送る
  (can be forwarded to another node)
    アルゴリズムの第三のケース
    (the third case in the algorithm)

  修復のためには (to repair)
  まず, 同じ行で, その桁が違うノードに問い合わせる
  (first, contacts the node referred to by another entry on the same row)
  そのようなノードがない場合, 次の行(もうひと桁一致の長い)に聞く
  (if it fails, contacts an entry on the next row)
    次の行は自分と l+1 桁一致しているので, そのノードの
    (nodes on the next row shares l+1 prefix with the local node)
    l 行目は自分と同じ l 桁の prefix を持つはず
    (on the l-th row of the contacted node may have R^{d}_{l})
    そこには探している R^{d}_{l} があるはず
    --
    論文では R^{i}_{l+1}, i!=d とあるが.
    l+1 桁目は関係ないのでは?

neighborhood set
  定期的にコンタクトして生存を確認しておく
  (contacts periodically to check if it is alive)
  応答のないノードがあったら
  その表の他のノードにneighborhood setを問い合わせ
  (if a node not responding, ask other members for their neighbor tables)
  その各ノードとの距離をはかって, 更新
  (check the distance of each of the newly discovered nodes and updates its table)

2.5 Locality
ここまでは basic routing properties
  routing hop と交換メッセージ数の最小化を問題にしてきた
  (we discussed performance in terms of the number of routing hops
   and the number of messages exchanged in join)

proximity metric の観点からも「よい」経路が得られる
(this section focuses on locality)

proximity metric
  スカラー (a scalar value, such as)
    IP でのホップ数 (number of IP routing hops)
    地理的な距離 (geographic distance)
    Round Trip Time
    ...
  距離を測定する手段はあると仮定
  (assumed that nodes can determine the distance)
    測定手段はアプリケーションによる
    traceroute, subnet maps, ...
  距離について, 三角不等式の成立を仮定
  (assume the triangulation inequality)
    "ac < ab + bc"
    IP でのホップ数では成立しない可能性も
    (this may not hold for some metrics, such as IP routing hops)
  三角不等式が成立しなくても, ルーティングは機能する
  (basic routing is not affected even if this does not hold)
    ローカリティに影響
    (will affect the locality)

新規ノード加入時の処理 (node arrival)

routing table での locality
  (復習)新規ノードXは物理的に近い既存ノードAに依頼して,
    XのID的な隣接ノード Z へメッセージを送信
    経路を X,A,B,...,Z として
    (the new node X send a join message by an existing node A,
     via B, C, ... to the node Z which is numerically closest to X)

  XはA,B,...,Zのrouting table の i行目を使用する
  (X uses the i-th row of the routing tables of A, B, ... Z)

  ローカリティとは
  routing table にあるノードが, 条件を満たすノードのうちで近い
  (locality means the nodes referred in the routing table
   is close to the present node)

  新規ノード加入前にローカリティがあるとして
  (assuming this property prior to the join of X,)
  ノードXが加入後も保たれることを示す
  (show how to maintain it after the join)

  A は X に物理的に近いものを選ぶ
  (A is requested to be close to X)
  - A のrouting tableの0行目は, どれもAに近い(仮定より)
    (from the assumption, nodes in the A's routing table is close to A)
  - A は Xに近い
    (and A is close to X)
  三角不等式によって, 0行目はXに比較的近い

(nodes in the 0-th row of A's table are close to X)

neighborhood set を A からもらうのも, 同様に正しい
(obtaining X's neighbor set from A is appropriate)

1行目はどうなるか (1-st row of routing table)
これは B からもらう (is obtained from B)
エントリは B からは近い(仮定より) (nodes in the row are close to B)
X からは近いか? (are they close to X?)
実はそれなりに近い (in reality, reasonably close to X)

表の各行の候補となるノード数は指数的に減少
(the number of candidates for a routing table entry is exponentially decreasing)
共通するプレフィクス長が長くなるのだから
(because they are required longer common prefix)
それらの平均距離は増加するはず
(average distance from B to the B1 (nodes referred by B's table) will increase)
ということは, X->A, A->B はそれに比べて小さい
(X->A and A->B is much smaller than that distance)
つまり, 1行目のエントリB1は B に近く, かつ X にも近いはず
(B1 is a reasonable choice for X1)

Figure 2
円周はそのレベルでの平均距離
(the circles around the n-th node indicates the average distance at level n)
表の相当する行のエントリは, そのあたりに存在

second stage
こうして作った routing table はそれなりによいが
(after X initialize the state)
累積する誤差を解消する必要
(this approximation must be improved)

X は routing table, neighborhood set の各ノードに表を要求
(X request the state from each of the nodes in its routing table and neighbor set)
それらの表の各エントリへの距離を測定
(compares the distance respectively, and updates if possible)
自分(X)の表をそれで更新 (もっと近いノードがあれば)

route locality
routing table のエントリは近いものを選んだ
ルーティングの各ステップは近いところに行く
(in each routing step, amessage is forwarded to a relatively close node)

しかし, ルーティングは大域的な情報(目標の方向とか)を使わない
(there is no sense of global direction)
最短経路が得られる保証はない
(no guarantee of the shortest path)

けれど, 相対的には良い経路が見つかる
(however, it results in relatively good routes)

○ A から B (距離d)へメッセージが転送されると, それ以降 A から d
以内のノードに転送されることはない
(if a message routed from A to B (distance d),
the message cannot be routed to C (within the distance d from A)
もしそういうノードがあったら, それは B を置き換えているはず
(Aのルーティングテーブルの B を置き換えているはず)
(B would be replaced by C if it exists)
○ステップ毎の転送距離は指数関数的に増加
(the distance of each step is exponentially increasing)
表の行ごとに, ノード数は指数的に減少するから
(because the number of possible nodes in a entry decreases exponentially)

以上二つの性質から, メッセージは戻ることなく目標に向かうはず
(the message has nowhere to go but towards its destination)

Figure 3

実験からのメッセージの軌跡の例
(sample trajectory of a message based on the experiment)
部分的に「あともどり」しても, 指数的な距離の増加で目的地に近付く
(the message may partly turn back during initial steps, ...)

Locating the nearest among k nodes
同じ対象データを, キーのIdに近い k 個のノードに複製する手法
(replicate information on the k Pastry nodes with the numerically closest nodeIds to the key)
例: PAST (a persistent storage utility)
ファイルの複製を作って, 高い可用性を実現
(replicate files to ensure high availability)

キーにId的に近いノードを探すのがルーティングだから,
k個のどれかには到達できる
(the message reaches one of the k nodes)

さらに, locality property によって
k個のうち, 出発点から距離的に近いノードに到達できる
(moreover, the message first reaches a node near the client)
ネットワーク負荷, レイテンシの向上に有益
(minimize latency and network load)

nodeId はランダムだから, k個は広く散らばる
(because the nodeId is assigned at random, they are widely dispersed)

Pastryのルーティングはの近いノードを探す
ステップ毎の距離はできるだけ小さくして
(in each step, the message travels the smallest possible distance)

これによって, k個のうち, 出発点から近いノードに最初に到達するはず
(a message tends to first reach a node near the client)

ただし, 近似に過ぎない (only approximation)
○ 大域的な方向などは使えないため (no global direction)
○ nodeIdが近くてもプレフィクスが違うケースが問題
(numerically near nodeId may not share prefix with the key)
たとえば, 0111 と 1000 など
最悪の場合, k/2-1 のレプリカはレベル0でプレフィクスが違う可能性
(in the worst case, k/2-1 nodes differ from the key in level 0)

そこで, heuristics
k 個のノードの近くまで来たら, ルーティングを切替え
(if a message approaches the set of k nodes, switches to numerically nearest address based routing)
アルゴリズムのケース3 (case 3 in the algorithm)
近くなったかどうかの判断は自分のrouting tableの密度から
(the decision of switch is based on the estimated node density)

この heuristic の効果: 3.3 の評価では
the nearest node - over 75%
one of the two nearest nodes - 91%