(Pastry つづき)

2.6 Arbitrary node failures and network partitions

arbitrary node failure
  ノードは反応はするが, 正しくなく振舞う/悪意の振舞い
  (continues to be responsive, but behaves incorrectly or maliciously)

これまでのrouting schemeは決定的
(routing scheme is deterministic)
  メッセージを受けて正しく転送しないノードがあるとだめ
  (repeated queries will fail if there is such node in the route)

それを許容するためには, routing をランダムにする
(the routing can be randomized)
転送先の条件を満たす複数のノードからランダムに選択
(randomly choice among multiple nodes that satisfy the criterion)
  best choiceを優先させるにしても
  (should be biased towards the best choice)

IP routing anomalies
  特定のホスト間だけ通信ができなくなる
  (a host is unreachable from only certain hosts)
nodeId 空間で隣接するノードが通信できれば動作するので, OK
(nodes are reachable as long as they are able to communicate with their
 immediate neighbors)

ただし, 修復によって, オーバーレイが分割されてしまう可能性
(may cause multiple, isolated Pastry overlay)
IP multicast を利用
  expanding ring multicast search で発見
  コストを減少させるために (to minimize the cost,)
    randomly, infrequently
    ホップ数の制限 (limited range of IP routing hops)
routing tableの改善にも役立つ
(the results can be used to improve the quality fo the routing tables)

3. Experimental results
prototype implementation
  Java
  simulated network
  up to 100,000 nodes
単一のJava VM内で動作 (run in a sigle Java VM)
proximity
  ノードを平面にランダムに配置
  (each node is assigned a location in a plane randomly)
  その距離を使う
    Internetは不均一だが
    (nodes in the Internet are not uniformly distributed)
    そうしたシミュレーションは実行中
    (currently performing emulations based on a more realistic topology)
    routing property に大きな影響はなさそう
    (not significantly affected)

3.1 Routing performance
(1)ホップ数とネットワークサイズの関係
   (number of hops as a function of the size of the Pastry network)
1000 から 100,000 ノードまで
  b=4 一桁のビット数 (number of bits in a digit)
  |L|=16 leaf set のサイズ
  |M|=32 neighborhood set のサイズ
ふたつのノードをランダムに選び, その間のメッセージのホップ数を測定
(two nodes are selected at random, and a message is routed between them)

Fig.4
  理論値は log_{2^{b}} N
  スケーラブル (show the scalability)

Fig.5
  ホップ数の分布 (N=100,000)
  (distribution of the number of hops)
  log_{16} 100,000 = 4.152410118609203 なので, 上限は 5

(2)経路のローカリティ
   (locality properties of Pastry routes)
平面距離での経路長を比較する
(compare the relative distance a message travels)
  Pastry routing
  完全なrouting tableがある場合
  (a ficticious routing scheme that maintains complete routing tables)
     1 hopで届く - つまりノード間の距離 - 下限

Fig.6
ficticious = 1.0
下限にくらべて 30% - 40% 長い
(only 30% - 40% longer)
テーブルサイズを考えると非常によい結果
(the result is quite good considering the small table size)

routing throughput
  3,000 messages per second
  非常に軽い (very lightweight)

3. 2 Maintaining the network
routing table の quality
  表のエントリのノードがどれくらい近いか

新規参加時の方式の違いの影響を見る
(three options to gather information when a node joins)
  新規参加時のメッセージの経由ノードから表を構成するが

○SL - 経由ノードの該当する行だけを利用
       (considers only the appropreate row from each node along the route)
○WT - 経由ノードの全情報を利用
       (use entire states, omitting the second stage)
○WTF - さらに, first stage の結果の表のノードから情報取得
        (second stage - fetch states from nodes appeard in the table after
         the first stage)

Fig.7
できあがった routing table について,
  縦軸: 最適な(最も近い)ノードがどれくらい見付かったか
  (vertical axis: quality of table)
    Optimal - 最適 (closest)
    Sub-optimal - 見付かったが最適ではない (not closest)
  横軸: 表の行の番号 (共通prefixの長さ)
  (horizontal axis: row index of the table - length of the common prefix)
level 2,3 は完全には埋まらない (ノード数が少ないから - 5000)
(level 2 and 3 are not fully populated)
WTFは非常に有効 (highly effective)
最適でないのは1エントリ以下
(less than 1 entry per level is not best)
SL,WTと比べて, 余分な情報交換(second stage)が効果をあげている

3. 3 Replica routing
キーに近いk個のノードに情報を複製して配置
(routing to one of the k closest nodes to a key)
それらのうち, 発信元に近いノードに到達できるか
(locate on of the k nodes near the client)
  k=5
  N=10,000
  b=3
  |L|=8

Fig.8
  縦軸: 到達したノードの順位の分布

(vertical: distribution of the rank of found node)
横軸: 新規参加方式 (SL, WT, WTF)
(horizontal: protocols for initializing a new node's state)
○Standard
without heuristic
○Estimation
with heuristic
○Perfect estimation
with heuristic
「近距離に来た」が正しく見積もれる(仮想的)
(the estimation of the approach is always correct)

heuristic (2.5)
十分近距離に来たら, Id的に近いノードを探す
(if a message approaches the set of k nodes, switches to
numerically nearest address based routing)

WTF, Standard
68%が最も近いノードに到達 (closest node)
87%が上位二つのどちらかに到達 (one of the two closest)
(Fig.8からは, 58%, 77%と読める?)

WTF, Estimated
76%, 92% に向上

SL, WTは良くない (strong negative effect)

Estimation と Perfect を比較しても, 2%程度しか悪くない
(heuristic is only 2% worse than the best possible)

3.4 Node failures
5000ノードのうち, 500(10%)をランダムにfailさせる
(500 nodes (10%) randomly fail silently)
そこでキー一つに対し, 二つのノードからメッセージ送出
(then two nodes send a message to a key)
これを100,000回くりかえし
(repeated 100,000 times)

つぎに, state repair をおこない
(next, the node state repair facility enabled)
さらに同一のメッセージを100,000回測定
(another 100,000 experiment for the same nodes and the same key)

Fig.9
repairによるrouting tableのquality
縦軸: routing table エントリの内訳
(vertical: routing table entry)
横軸: レベル, {fail前,fail後,修復後}
(horizontal: level of routing table)
修復はそのエントリを参照した時に行なう(lazy)
(repaired lazily - when they are being used)
アクセスされなかった表は修復されない
(entry not used is not repaired)
これはデータからは除外してある
(they are excluded)

結果 (result)
missingをすべて修復した
(all missing entry is repaired)
optimalの割合もfail前に近い
(quality approaches that before the failure)

ただ, レベル0のoptimalの割合が低い
(for level 0, optimal is about 1 low)
sub-optimalとoptimalの差は小さかった
(actual distance between the optimal and the subobtimal is very small)
レベル0はもともと近距離だから
(entries in level 0 row are close to the current node)

  emptyの増加はノード数の減少のため
  (in level 1, 2, the increase of empty is due to the reduction of nodes)
    修復失敗ではない
    (not reduction of the quality)

Fig.10
fail, repairのroutingへの影響
  縦軸: 平均ホップ数
  (vertical: average hops)
  横軸: fail前,fail後,修復後
  (horizontal: before, after the fail, after the repair)
修復で, fail前よりすこし高いだけに戻っている
(after the repair, is only slightly higher)

修復のコスト(メッセージ数)
(the cost of the repair - number of messages)
  1ノードのfailの影響を修復するのに, 平均57回のメッセージ
  (57 messages to repair all entries per failed node)

4. Related works

Tapestry: A Resilient Global-scale Overlay for Service Deployment
Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph and
John D. Kubiatowicz
(IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 22, NO. 1, JANUARY 2004)
アルゴリズムの詳細については,
Distributed Object Location in a Dynamic Network
Kirsten Hildrum, John D. Kubiatowicz, Satish Rao and Ben Y. Zhao
(SPAA ’02)

DOLR
  decentralized object location and routing

III. Tapestry Algorithms
A. The DOLR Networking API
node
  nodeID
  N_{id} -- latex ふうの下つき文字の指定
object
  GUID (globally unique identifier)
  O_{G}

node, object どちらも同じ空間から
  現状は160bit
  SHA-1 (hash function)

複数のアプリケーションで overlay を共用
(share an overlay among many applications)
  区別のために application-specific identifier A_{id}
    TCPにおけるポート番号みたいなもの
    (resembles a port number in TCP/IP)

基本的API
  PublishObject(O_{G}, A_{id})
  UnpublishObject(O_{G}, A_{id})
  RouteToObject(O_{G}, A_{id})
  RouteToNode(N, A_{id}, Exact)

B. Routing and Object Location
(1)routing mesh
  identifier G -> unique live node G_{R} -- the identifier’s root
  もし N_{id} = G なら, それが root

routing table
  "neighbor map"
  Plaxton の方式 (つまり Pastry とおなじ)

level (j+1) <- 自分と共通する prefix 長 = j
(the length of the common prefix is j)
i-th entry <- その次の桁が i
(the next digit is i)

エントリには closest node が入る
locality を提供

Fig.1
(SPAA02)
Property 1 (consistency)
表に空きがあるのは, 該当ノードが全システム内に存在しない場合だけ
(a table entry can be null only if there is no such node in the system)
Property 2 (locality)
表にあるノードは「最も近い」ノード
(referred node in a entry is the closest node)
Property 3 (unique root set)
あるオブジェクトに対する root set は, ネットワークのどこで
求めても同じものになる
(the root set for a object must be unique)

Fig.2
example of routing
表から, 次の桁が一致するノードへ転送
(forward message to the node with next matching digit)

ホップ数は log_{beta} N
N - namespace size
beta - ひと桁の大きさ (the base of the digit)

surrogate routing
idに対応するノードがない場合
(if the table entry is null)
"近い"ノードへルーティング
(looks for a "close" digit in the table)

Fig.3
メッセージの転送先を決める
(locates the next hop towards the root)

NextHop(n,G)
G - 目標のID (destination GUID)
n - 何桁目 (tne digit position in the id)
L.1 - 最後のレベルに到達 - 自分がゴール
(all digits matched - the destination is myself)

L.4 - d: Gのn桁目; e: 表の対応するノード(n行 d列)
(d <- n-th digit on G; e <- entry in the table)

L.5 - 対応するノードがないなら
(if the entry is empty)

L.6 - その行の右をみていく (d = d + 1)
その行には少なくとも自分は含まれている
(at least myself is included in this level)
(この部分がsurrogate)

L.9 - 見つかったのが自分だったら
(if the result is myself)
L.10 - 次の桁も自分でやる
(try the next digit by myself)
L.11 - そうでないなら (otherwise)
L.12 - 転送先は e (forward the message to e)

動的な状況への対応 (in a dynamic network environment)
リンクの変化や故障
(link change or failure)

表の各エントリに複数(c個)のノードを入れておく
(backup links - redundant routing paths)
さらに, 逆向きのポインタも保持
(reverse reference to nodes that point at it)

(2)object publication and location
server S - object(id=O_{G}) を保持するノード
(store the object)
これはそのオブジェクトのroot(O_{R})とは別
(different from the root node of O)

server S は定期的に publish message を送信
(S route a publish message to the root periodically)
宛先のidは O_{G}
これは root node に転送される

この publish message を中継したノードは
(each node along the publication path stores <O_{G}, S>)
<O_{G}, S> を記録する

objectの複製が複数あった場合には
(when there are replicas on separate servers)
その所有ノードがそれぞれ上記を実行
(each server publishes its copy)

Fig.4
server: 4228, AA93
root: 4377
dotted line points the server

その object を探す client は
(clients locate O by routing a message to the root of O)
O_{G} に向けてメッセージを送信
最終的には root に到達するが
途中で <O_{G}, S> を保持するノードに出会うと
(if a node along the path holds <O_{G}, S>)
ただちに所有者が判明する
(the message will be forwarded to the server S immediately)
rootまで届く前に
(before it reaches the root)
しかも, 複製が多数ある時は, 近くのものが最初にみつかる
(queries are routed to nearby object replicas)

Fig.5
object: 4378
clients: 4664, 4B4F, 57EC

C. Dynamic Node Algorithms
1) 新規ノード N が参加 (Node Insertion)

やるべきことは4つ
a) 他ノードの表で, N が入るべき空所があれば通知
(N fills a null entry in some nodes)
b) N が既存のオブジェクトの root になるなら, その参照を移動
(move references to objects whose root is now N)
c) N のための(ほぼoptimalな)表を作成
(construct a near-optimal routing table for N)
d) N の近くのノードに通知して, それらの表の改善
(notify nearby nodes to optimize their tables)

(1)まず, N の id (N_{id})を自分で計算し
(N computes its id, N_{id})
初期ノードに依頼して, N_{id} 宛のメッセージを routing
(N routes a message to N_{id})
S (N’s surrogate) が見つかる
(N’s surrogate is reached)

S の id と, N_{id} の共通 prefix C の長さを p として
(p: length of the common prefix between S_{id} and N_{id})
全ノードのうち, C ではじまるすべてのノードに通信 ★
(S sends a message to all nodes sharing the same prefix (length p))
  a), b) をすませる
  (completes a) and b))

★をどうやって行なうか
Acknowledged Multicast(alpha, func)
  Figure 8 (SPAA '02)
  alpha - prefix
  func - prefix にマッチするのが自分一人だったら実行する関数
    (apply this function if the match is only one)
  ノードの routing table を利用
  (use the routing table)
  alpha ではじまる次のレベルのエントリに再帰的に転送
  (forward the message recursively to nodes whose prefix is alpha)
    一つだけでOK
    自分も含む (including the local node)

  もし, 一つのエントリに該当するノードが複数あったらどうなるか
  いずれどこかのレベルで分岐するはずだから大丈夫