ネーミング（1）

分散システム

2013年12月2日

建部修見

ネーミング

- 資源の共有、実体の識別、位置の参照
- 名前の解決（**Name Resolution**）＝参照している実体に解決
- **ネーミングシステム、リソルバ（Resolver）**
- 分散システムで利用される名前
  - ヒューマンフレンドリな名前
    - パス名、URL
  - 位置に依存しない名前（フラットな名前）
    - ハッシュ値、移動体の参照
  - 属性で指定される名前
    - 属性により検索

名前、識別子、アドレス

- アドレス（例：IPアドレス）
  - 実体をアクセスするためのアクセスポイントの名前
  - 名前の一種であるが、実体に強くバインドされている
    - 柔軟性がない（負荷分散などに利用しにくい）
    - ヒューマンフレンドリではない
- 識別子（例：MACアドレス）
  - 実体をユニークに参照する名前、参照は変わらない
    - マシンリーダブルなビット
- ヒューマンフレンドリな名前
  - パス名、DNS名

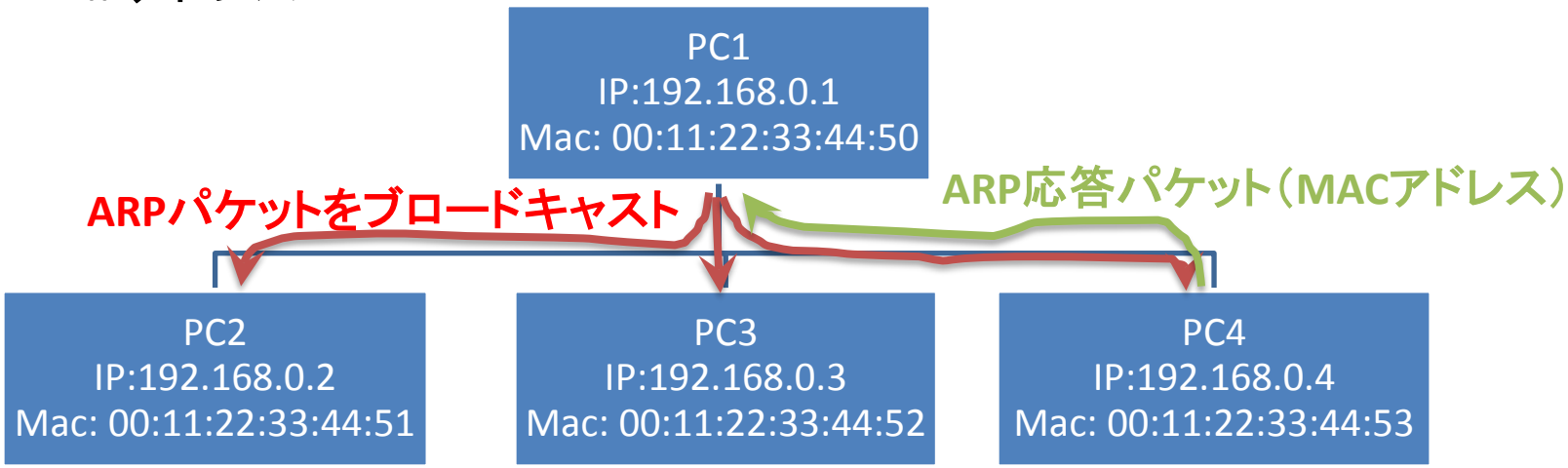
ネーミングシステム＝名前→アドレスの変換

フラットな名前の解決

- フラットな名前
  - Locateするための情報を持たない
  - 実体をユニークに参照する識別子
- フラットな名前に対する解決法
  - ブロードキャスト、マルチキャスト
  - Forwarding Pointers
  - 分散ハッシュ表（DHT）

ブロードキャスト（1）

- Internet Address Resolution Protocol (**ARP**)
  - 宛先IPアドレスのEthernetアドレス（MACアドレス）を知る
  - ARPパケットをブロードキャスト
    - 送信元MACアドレス，送信元IPアドレス，宛先IPアドレス
  - 該当IPアドレスのサーバはARP応答パケットにMACアドレスをユニキャストで返送
    - 宛先MAC アドレス，宛先IPアドレス，送信元MACアドレス，送信元IPアドレス



ブロードキャスト（2）

- Wake on LAN（WOL）
- 電源の入っていないマシンにマジックパケットを送付するため、ブロードキャスト
  - 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
    MACアドレス x 16
- ブロードキャストはLAN（L2ネットワーク）で利用可能
  - See ifconfig(8) in Unix or ipconfig /all in Windows

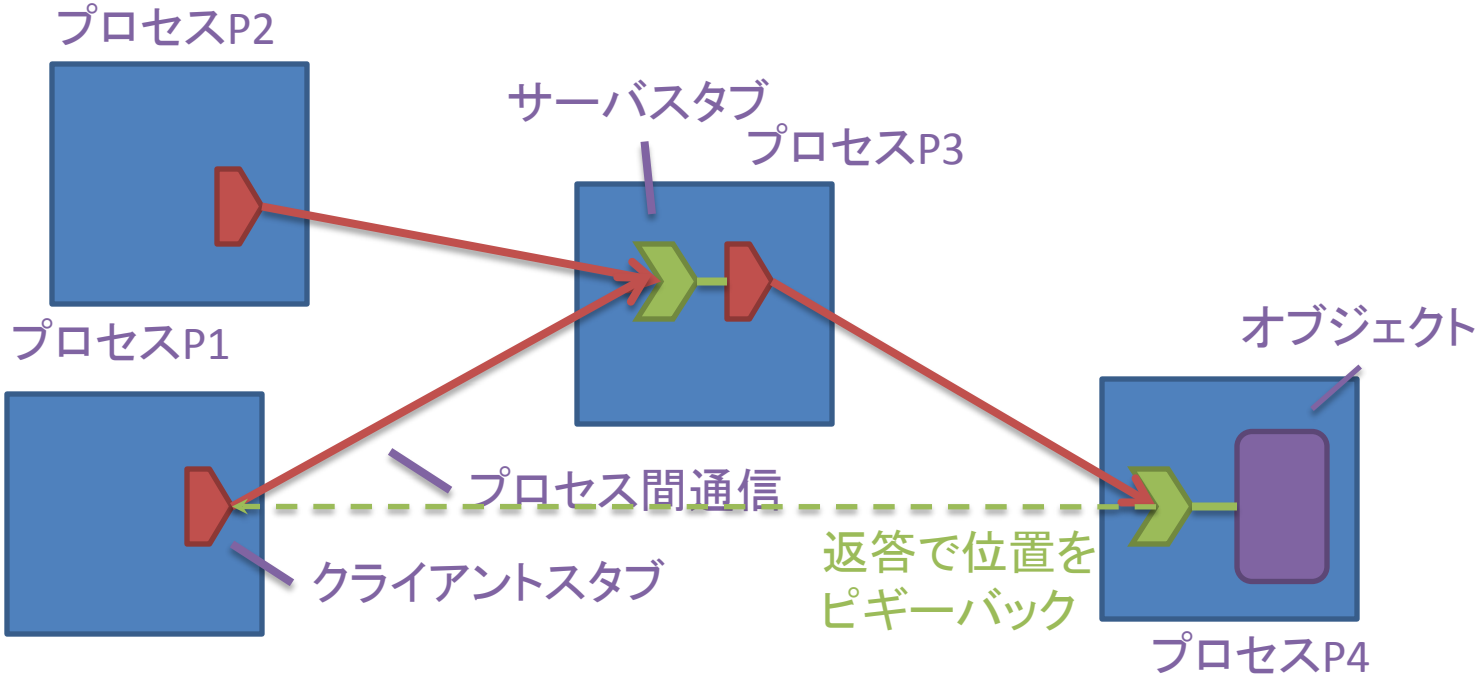
```
% ifconfig
eth0   Link encap:Ethernet  HWaddr 00:11:22:33:44:55
       inet addr:130.158.109.100  Bcast:130.158.109.255  Mask:255.255.255.0
```

- スケーラブルではない
  - 必要以上のホストが割込まれる
  - ブロードキャストはネットワークを簡単に過負荷にできる（要注意！）
    - ネットワークは共有資源
    - 譲り合いの精神が大事
- マルチキャストはすこしまし

```
struct socketaddr_in addr;                          // socket address for Internet protocol
struct hostent *hp = gethostbyname(host);           // host should be broadcast address
char mpacket[6 + 6 * 16];                           // magic packet
int val = 1;

if (hp == NULL || hp->h_addrtype != AF_INET)
        return (-1);

memset(&addr, 0, sizeof(addr));                                  // zero clear
addr.sin_port = htons(9);                                        // port 9 in network order
addr.sin_family = hp->addrtype;                                  // AF_INET
memcpy(&addr.sin_addr, hp->h_addr, sizeof(addr.sin_addr)); // copy IP address

sock = socket(PF_INET, SOCK_DGRAM, 0);                           // UDP socket
if (sock == -1)
        return (-1);                                             // set broadcast socket flag
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &val, sizeof(val));

memset(mpacket, 0xff, 6);                  // 0xff 0xff 0xff 0xff 0xff 0xff
for (i = 0; i < 16; i++)                   // copy mac address 16 times
        memcpy(mpacket + 6 + i * 6, mac, 6);
                                                                 // broadcast a magic packet
rv = sendto(sock, mpacket, sizeof(mpacket), 0, (struct sockaddr *)&addr, sizeof(addr));
return (rv);
```

Forwarding Pointers（1）

- 移動体の位置を確認する方法
- AからBに移動したとき、AにBの参照を残す
- 利点：単純、クライアント透明に移動可能



Forwarding Pointers（２）

- 欠点
  - リンクが長くなりすぎる
  - 全ての中間地点でForwarding Pointersを保持する必要がある
  - リンクが途絶える可能性がある
- リンクを短く、頑強にする必要がある
  - リンクのショートカットの作成
    - 到達したオブジェクトがクライアントに返答を返すときにオブジェクトの位置をピギーバック
    - 参照されなくなったサーバスタブの分散ガーベジコレクション（分散GC）
  - ホームで最新の位置を冗長に保持

分散ハッシュ表（DHT）

- 様々なシステムが存在：Chord, Pastry, Tapestry, CAN, Kademlia, . . .
- Chord
  - ノードやキーの識別子は*m*ビットID空間にランダムに割当
    - 128ビット（MD5），160ビット（SHA1）
  - *m*ビットID空間を分割し複数ノードで管理
    - キー*k*の値は、最小の*id*（≥*k*）を持つノードが保持
    - *id = succ(k)*
- **キー*k*から*succ(k)*のアドレス解決**（＝キーkを管理しているノード検索）を効率的に行うことが鍵！

フィンガー表

- Chordの各ノードが保持するmエントリのルーティング表

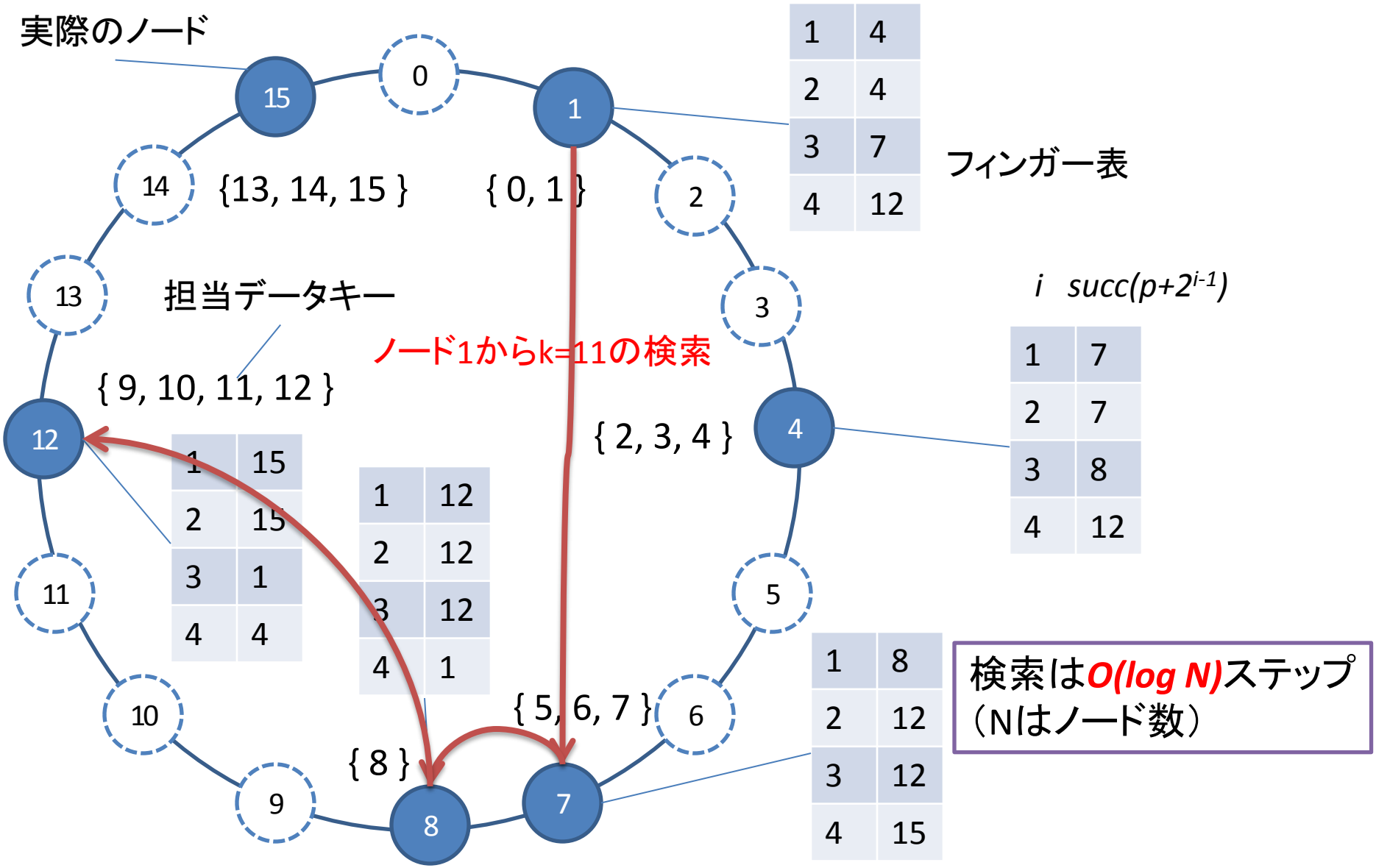
  $FT_p[i] = succ(p + 2^{i-1})$
- 2^{i-1}先の*successor*を保持
  - 2^{i-1}先へのショートカット
- キーkの検索は

  $q = FT_p[j] \leq k < FT_p[j + 1]$

  (or $q = FT_p[1], p < k < FT_p[1]$のとき)

  を満たすj番目のエントリqに検索リクエストをフォワード

ノード1からkey 11の検索



ノードの参加、脱退

- フィンガー表に加え一つ前のノードを指すpredecessorも管理する
- ノード*p*が参加
  - 任意ノードで*succ(p+1)*を検索
  - *succ(p+1)*のpredecessorに*p*を設定
  - *p*の$FT_p[1]$に*succ(p+1)*を設定
- ノード*p*が脱退
  - *succ(p+1)*のpredecessorをunknownに設定

フィンガー表の更新

- 各ノードqでは$FT_q[1]$が最も重要。定期的に以下を実行し更新
  - *succ(q+1)*にpredecessorを返してもらう
  - qであればOK
  - 異なれば$FT_q[1]$を更新
  - （Unknownの場合、*succ(q+1)*のpredecessorをqに更新）
- $FT_q[i]$の更新は定期的に$succ(q+2^{i-1})$を検索
- 詳細はStoica et al (2003)を参照

ネットワーク近接性

- オーバレイネットワーク一般の問題
  - Chordのフィンガー表でたどると大陸を何度もまたがって検索してしまう、など
- 三つの解決法
  - トポロジベースのID割当
  - 近接ルーティング
    - Chordの場合、フィンガー表の各エントリに*r*ノード保持
    - $FT_p[i]$に$[p+2^{i-1}, p+2^i-1]$のエントリを冗長に保持
  - 近接ノード選択
    - Pastryなど、ルーティング表のエントリを選択可能な場合、ルーティング表に近接ノードのエントリをいれる

まとめ

- ネーミングシステム
- フラットな名前の解決
  - ブロードキャスト
  - Forwarding Pointers
  - 分散ハッシュ表（DHT）