cloud 研究会 @マイクロソフト(株) 新宿本社
2008年 12月 19日(金)

scale out の技術

～ consistent hashing 編

首藤 一幸

contents

- 導入: scale out の方策
- consistent hashing
- consistent hashing の応用
  - サーバ側の技術
    オンメモリの key-value store
  - peer-to-peer 由来の技術
    structured overlays / 構造化オーバレイ
  - ２つの関係
- Azure Services Platform と構造化オーバレイ

首藤一幸 (35)

- 研究職 → スタートアップ → 大学
  - 1998年 早稲田の助手
  - 2001年 産業技術総合研究所
  - 2006年 ウタゴエ 取締役CTO
  - 2008年 12月 東京工業大学

- ソフトウェアエンジニア, 技術フェチ
- 技術者、研究者として
  - 分散システム, オーバレイネットワーク, peer-to-peer, プログラミング言語処理系, 高性能計算
- わかりやすいところで
  - Java JITコンパイラ shuJIT (1998〜)
  - 書籍 Binary Hacks (2006.11) 執筆
  - peer-to-peer ライブ配信ソフト (2006〜)
    - 未踏ソフトでスーパークリエータ認定, ウタゴエで商用化
  - カラオケグリッド: 全世界カラオケセッション (2003.11)
    - Access Grid を使って 5ヶ国, 2X拠点で。
  - AES候補暗号方式の高速な実装 with NTT (1998)

scale up vs. scale out

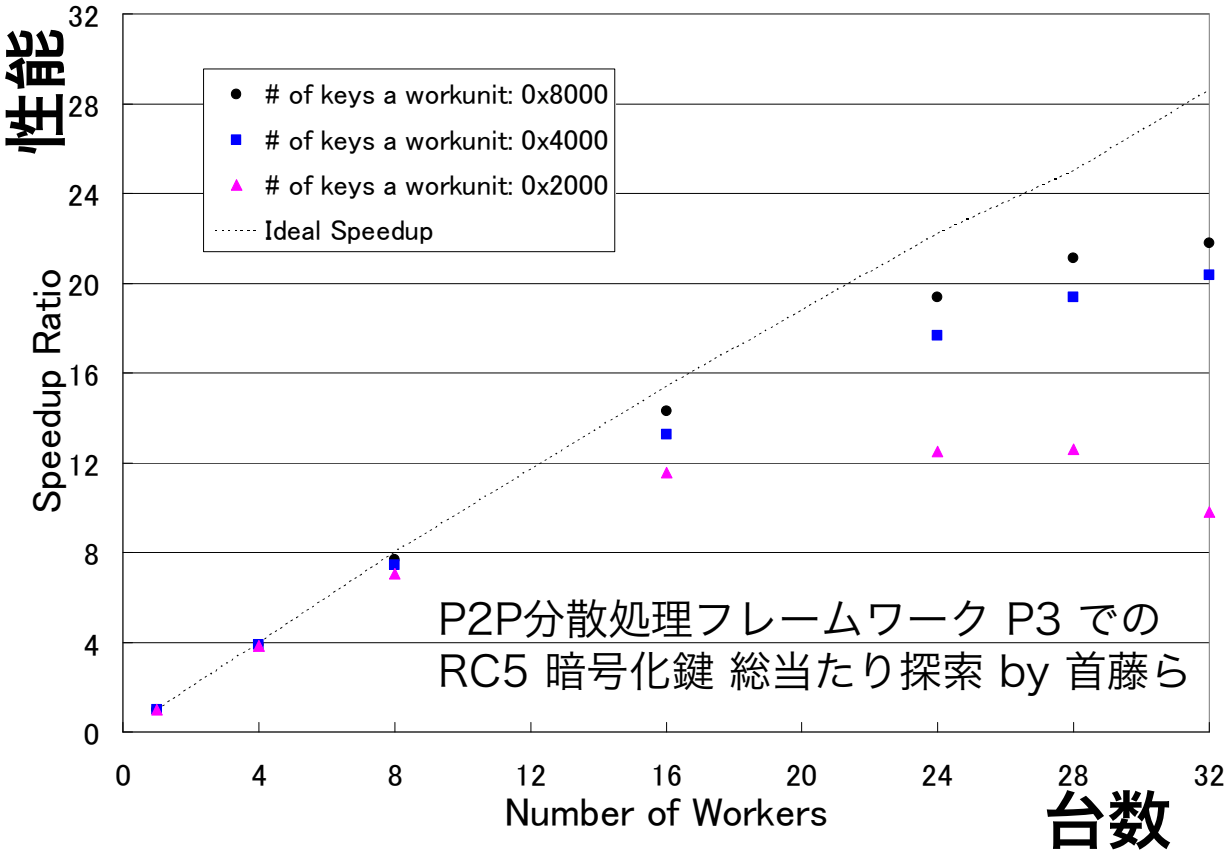
- scale up vs. scale out　性能向上のために
  - scale up　1ノードを強化すること
  - scale out　ノード数 (台数) を増やすこと

- コストパフォーマンス
  - ×　scale up 例: 2倍の性能のために 10倍のお値段
  - ◎?　scale out (うまくやれば) お値段に比例した性能

- うまくやって、
  お値段に比例した性能を享受したい。
  - 友人の親父「10台で 10倍より早くするのがプロだろ！」

scale out させたい！

- が、なかなか、そうはいかない。
  - Linear speedup は困難。

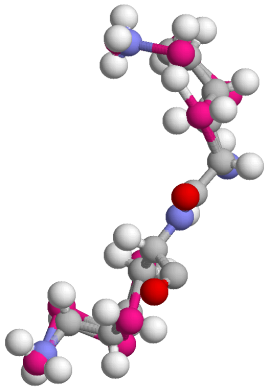


- これは
  マスタ・ワーカ型
  の並列処理
  - linear speedup させやすい方。
  - 計算の合間に通信が必要な並列処理だと、もっと悲惨。

scale out のためにはそれなりの方策が要るということ

scale out させるための方策
ノード数に比例した性能 (に近い性能) を得るための方策

- 問題を選ぶ
  - フィットしない問題はあきらめる。
    - スパコンでの数値計算を cloud で…
      スパコンは、メモリ帯域幅とノード間の低遅延通信が命。



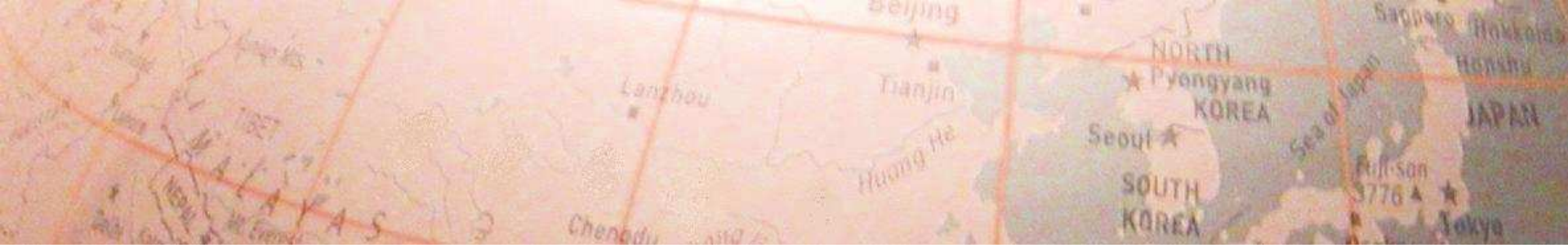
- **解き方**を選ぶ
  - アプリでの工夫
    - 例: タンパク質の立体構造決定
      スパコン (決定論的手法) vs. Folding@home (統計的手法)
  - システム側での工夫　→ 汎用性がそれなりに高い手法
    - 例: 全体をロック vs. 部分々々をロック vs. lock-freeアルゴリズム
      - Nプロセッサで、N倍になるべく近い性能を得るための方策。
      - これはマルチプロセッサでの話だが、分散でも同様の話がある。
    - フィットする問題や解き方を規定することもある。
      - 例: Azure SDS では、Partition Key を指定しないクエリは全ノードに…

scale out させるための方策

汎用性が それなりに 高いもの

- ブロードキャスト, マルチキャスト
  - flooding → gossip → 枝の刈り取り → 配送木を構築
- pure P2P でのデータ lookup
  - flooding → key-based routing (構造化オーバレイ)
- データを分担するノードの決め方
  - mod N → consistent hashing
- key-value store上の構造化データ / 表の組み立て方, 使い方
- 一貫性保証
  - 全体をロック → 部分々々をロック → lock-freeアルゴリズム
  - ロック → transactional memory (投機的実行 & コミット: 検証 + 反映)
  - ユーザによる保証 / 不整合解決のサポート
    - vector clocks, …
  - 一貫性モデル: どういう保証がなされるかの宣言
    - ○○ consistency
  - (分散) 合意プロトコル
    - Paxos, …

work in progress



consistent hashing

データを分担するノードを決める方法

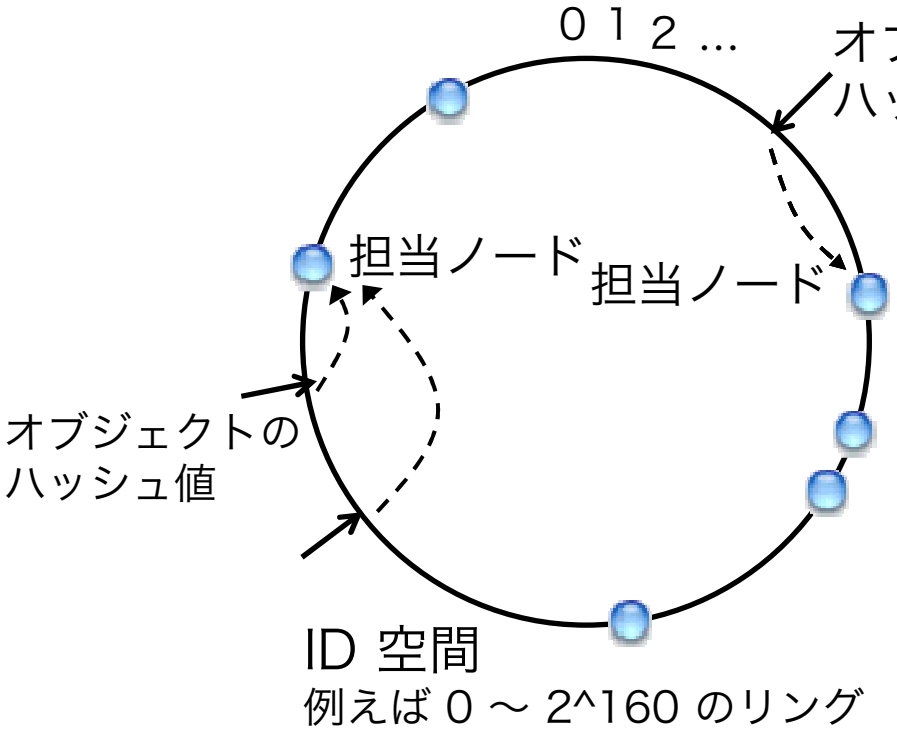
- オブジェクト o をノード n台のどれかに持たせる
  - 例: **ウェブのキャッシュサーバを複数台で構築**
    オブジェクト = ウェブコンテンツのファイル

- よくある方法
  - hash(o) mod n = 担当ノード番号 として決める
  - ノードの追加、除去を行うと…
    - ほとんどすべてのオブジェクトの担当ノードが変わってしまう！
      - 例: 17 mod 3(台) = 2(番ノードが担当) だったのが 17 mod 4 = 1 に変わる。
    - 保持しているほぼ全てのオブジェクトについて、担当ノードが変わる。
      再配分 → 大量の通信が必要。
    - さもなくば
      - 再配分しない → オブジェクトを取得できない。
        ウェブキャッシュが役立たずに…
      - n を変更しない → ノード追加は無意味となり,
        ノード除去ではキャッシュ不能オブジェクトが出てくる。

- ノード数が増えるほど、非現実的になっていく。
  - 1台でも故障すると、ほぼ全オブジェクトの再配分が必要に。

データを分担するノードを決める方法

- そこで consistent hashing
  - ノードの追加・除去によって再配分される
    オブジェクトが 1/n で済むような分配方法。



- ノードとオブジェクト、
  両方に ID を振る。
  0 ～ 2^64 とか 2^160 とか

- Chord の場合 (図)
  - 時計回りで、最初に
    ぶつかったノード
- Pastry 等
  - ID が数値的に最も
    近いノード

consistent hashing の応用

- **サーバ側**
  オンメモリ key-value store (の一部)
  - memcached
    - クライアントライブラリ次第
  - Dynamo (Amazon)
  - ROMA (楽天技術研究所)

  - 使い方
    - RDB から得たデータのキャッシュ。
    - 有効期間が短い (volatile) データの保持。
      - e.g. HTTPやSIPセッション情報, 座席予約, 株式・債権取引, 最終ログイン時刻, …

性能重視

no-hop が多い

～ 1,000台？

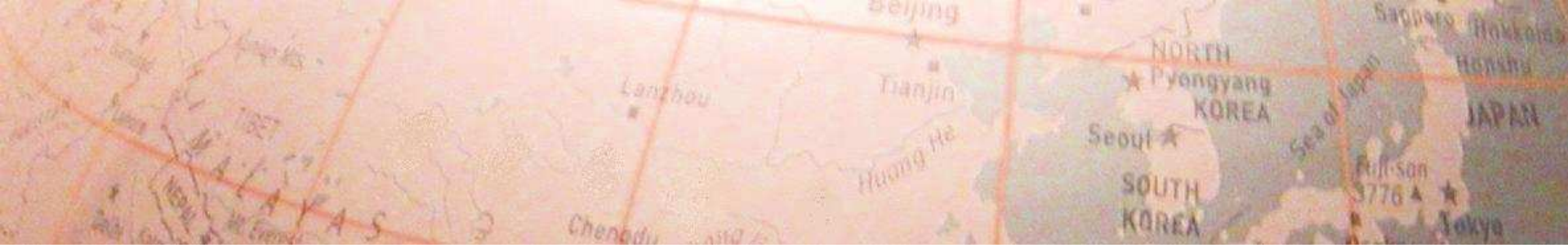
経路表の大きさ・保守コストと経路長のトレードオフ

～ 数百万台？

スケーラビリティ重視

- **Peer-to-peer 由来**
  構造化オーバレイ / 分散ハッシュ表
  - Chord, Pastry, Kademlia, …
  - Azure の基盤に両方向版 Chord が使われる?

multi-hop の routing / forwardingをいとわない



サーバ側の技術

(オンメモリの) key-value store

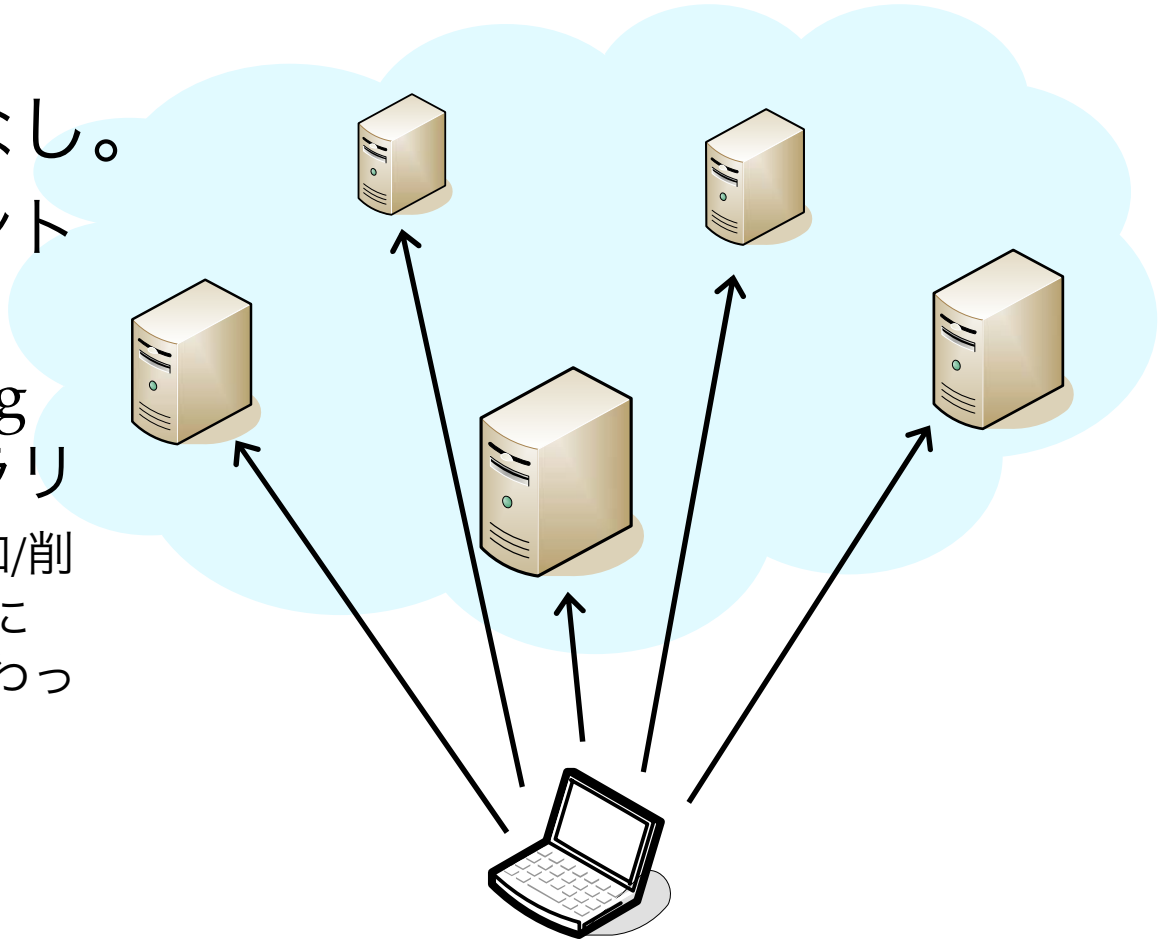
with consistent hashing

memcached

- Brad Fitzpatrick氏が、LiveJournal (blogのhosting) のために開発。
- 以後、ウェブ上サービスの裏側で広く使われている。
  - はてな, livedoor, mixi, Vox, Facebook, YouTube, Digg, Twitter, Wikipedia, …
- 規模の実績
  - mixi が 100台以上で運用。
  - Facebook が 800台以上で運用。1.2.3 ベースの改造バージョン。
- 特徴
  - テキストベースのシンプルなプロトコル。
    - set <key> <flags> <exptimes> <bytes> [noreply]¥r¥nデータ¥r¥n
    - get <key>*¥r¥n
    - このプロトコルを採用するソフトも多い: Tokyo Tylant, Overlay Weaver, ...
    - 1.3 にはバイナリプロトコルも入る。少し速いらしい。
  - 性能重視の実装。数万 request/sec 以上。
  - サーバ群は連携せず、担当ノードはクライアント (ライブラリ) が決める。

memcached

- 担当ノードはクライアントが決める。サーバ群は連携なし。
  - 方式はクライアントライブラリ次第。
  - consistent hashing ではないライブラリもある → ノード追加/削除によって、キー全体に渡って担当ノードが変わってしまう。



Amazon's Dynamo

- 論文が出ている。SOSP'07。
  - 新奇なアイディアの提案ではなく、手堅い設計。

- Amazon の内部で使われている。
  - ショッピングカート。
  - Amazon Web Services (AWS) の SimpleDB は、おそらく Dynamo で運用されている。
    - SimpleDB
      - 2007年12月、β版公開。
      - データモデル: domain に item が入り、item は attribute-value ペアで表現される。スプレッドシートに例えると、domainはワークシート、attributeは列のヘッダ、valueはセルに入ったデータ。
      - API
        - » void PutAttribute(domain, item, 複数のattribute-valueペア)
        - » 複数の attribute-valueペア GetAttribute(domain, item)

amazon web services™

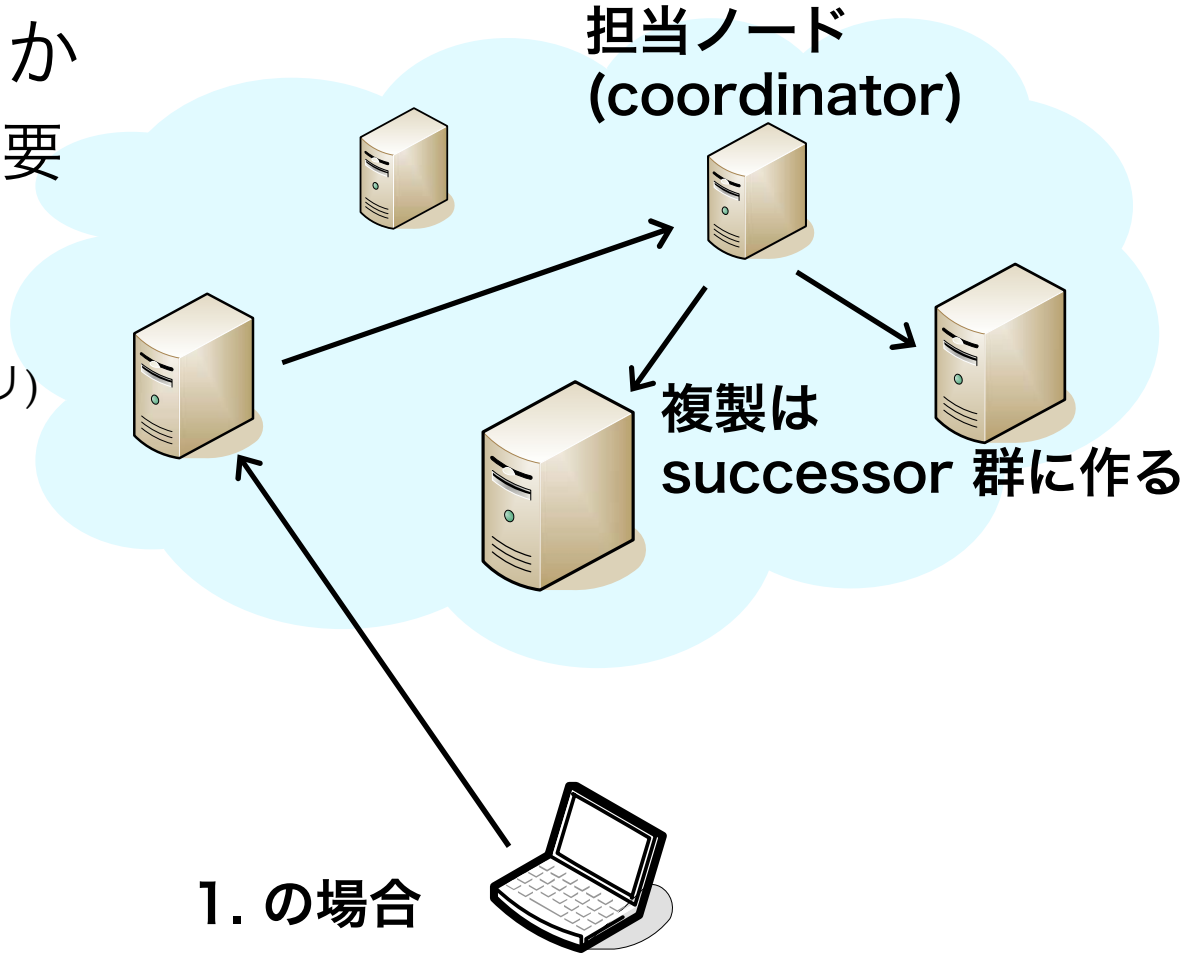
Amazon EC2
Amazon S3
Amazon SimpleDB
Amazon SQS

Amazon's Dynamo

- 狙う規模
  - 数百台
- 担当ノードの決め方
  - consistent hashing
- 一貫性
  - eventual consistency
  - 不整合が起きた場合、データに付いてるvector clocks [Lamport1978] を手がかりに、アプリ側で解決する。
- Service Level Agreements (SLA)
  - ウェブアプリのバックエンド → 遅延の低さ重視。
  - クライアントとの SLA に基づいて、パラメータ (e.g. 複製数) を調整する。
    - SLAの例: 500 req/s までなら要求の 99.9 % までは 300 msec 以内に返答を返す。

Amazon's Dynamo

- 担当ノードを決めるのは、次のどちらか
  1. クライアントから要求を受けたノード
  2. クライアント
     (要 クライアントライブラリ)



楽天 ROMA

- Rakuten On-Memory Architecture
  - 「複数マシンから構成されるオンメモリストレージ」
  - Ruby で実装。
- 2007年開発開始、2008年 9月社内向けα版。
- 狙い
  - 高い耐故障性 → 複製
  - それなりのパフォーマンス (耐故障性重視)
  - 高いスケーラビリティ
    - 狙いは数百台
    - 2008年 6月時点で「6台で 10ノードが動いてて…」by まつもとさん

楽天 ROMA

- 「ROMAの出発点をConsistent Hashingとする。」

ROMA への道（その２）

発想から基本設計へ

2007/09/01

楽天技術研究所　鳥居　順次

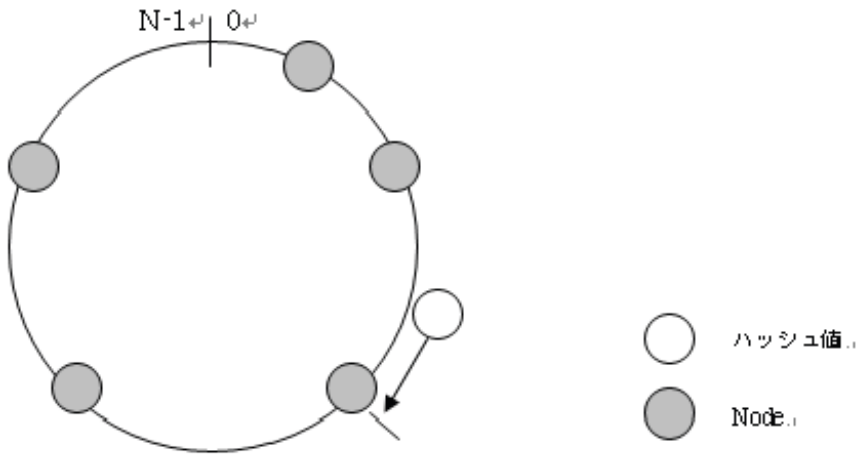
▪ これまでの道のり

ノードは環状に配置する方向で間違えはなさそう。

環状ハッシュは既に P2P の分野で大きな成功を収めている。広く知られている Chord 等の分散ハッシュのアルゴリズムを参考に ROMA を目指すゾ。

▪ ハッシュ関数とノード

ROMA の出発点を Consistent Hashing とする。ハッシュ関数は SHA-1 を想定し巨大な空間（最大 2^{160}）を定義する。ハッシュ値に対する担当範囲は時計回りに回ったときに到達する最初のノードとする。



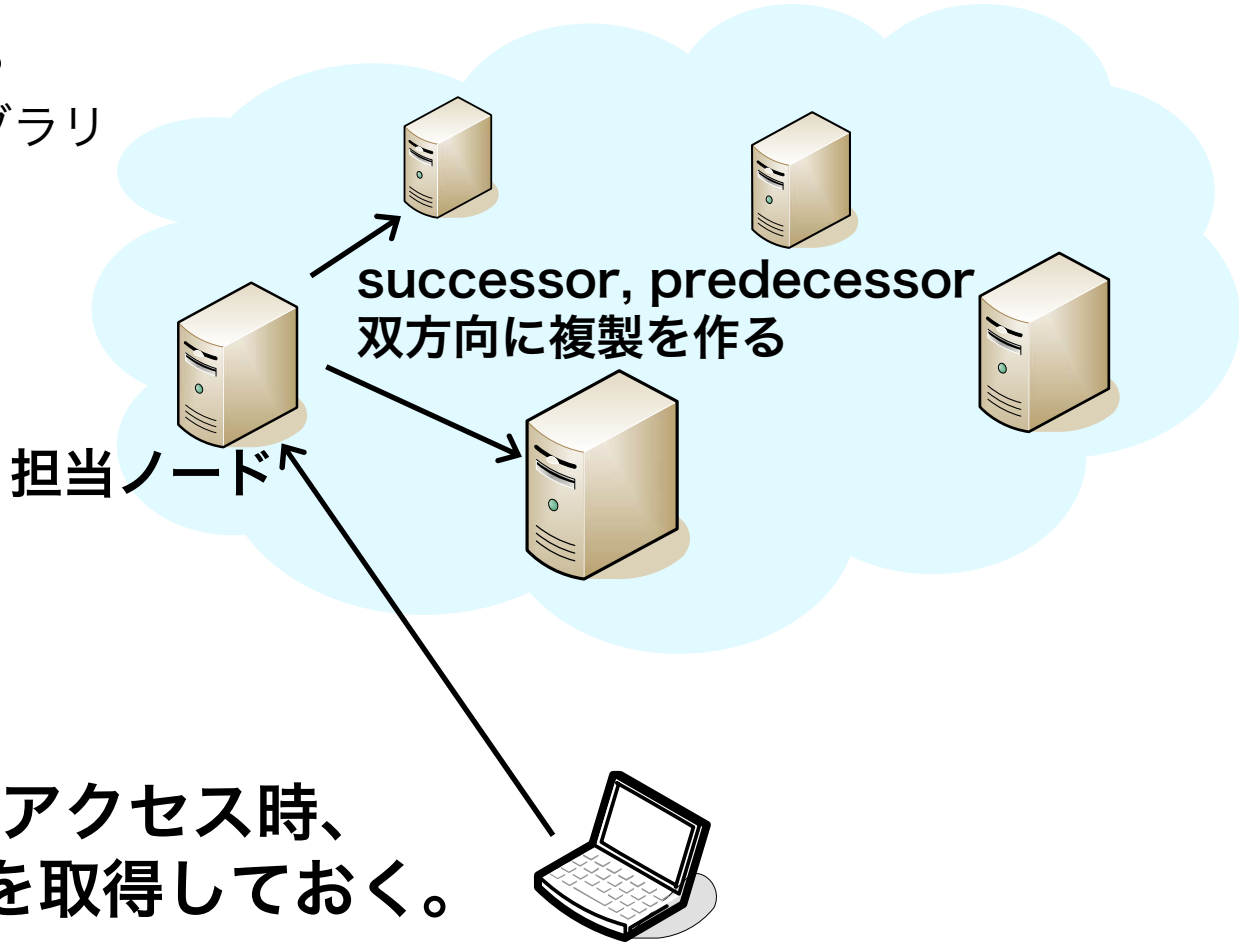
楽天技術研究所の紹介スライドより 2007年9月時点の設計メモ

楽天 ROMA

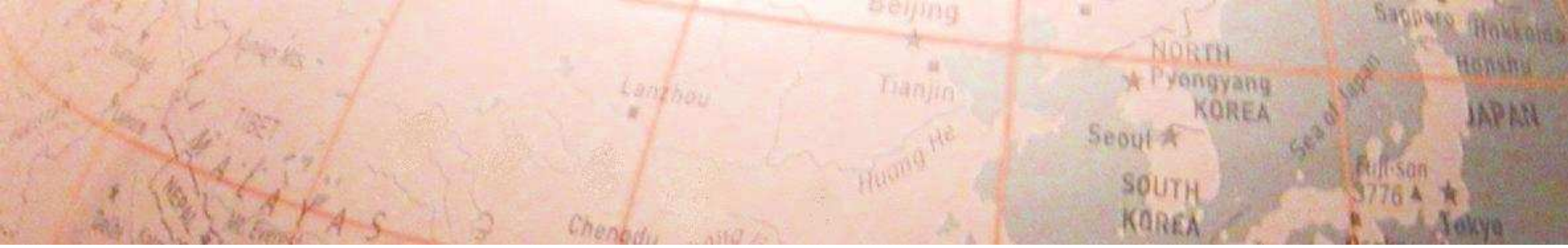
2008年 10月、クラウド研究会での西澤氏

楽天 ROMA

- 担当ノードはクライアントが決める。
  - 要クライアントライブラリ



ROMAへの初回アクセス時、
全ノードの情報を取得しておく。

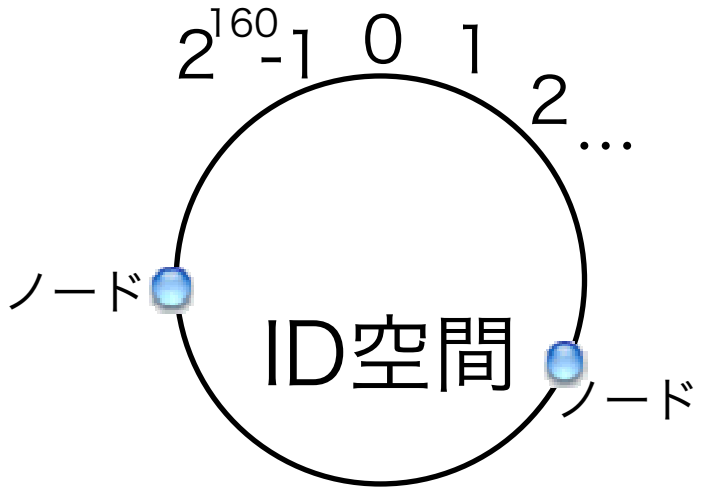


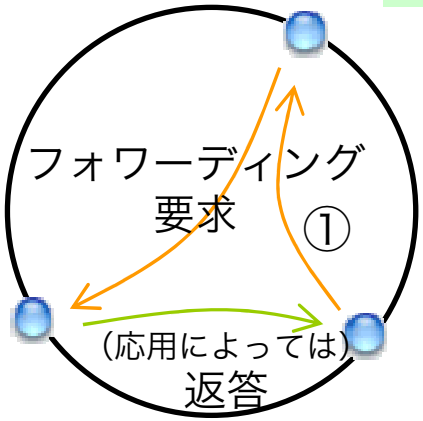
peer-to-peer 由来の技術

structured overlays / 構造化オーバレイ

構造化オーバレイの基本

- ノード（計算機）とオブジェクトの両方にIDが振られる。
  - IDはたいてい整数値。160ビットだったり 128ビットだったり。
  - オブジェクト：任意の文字列だったり、ファイルだったり、プロセスだったり。
- ノードは、ID空間中のある範囲を受け持つ。
  - だいたい、ノードのIDと数値的に近い範囲を受け持つ。
- IDを宛先として**ルーティング/フォワーディング**が行われ、その行き着く先は、そのIDを受け持つ**担当ノード**となる。





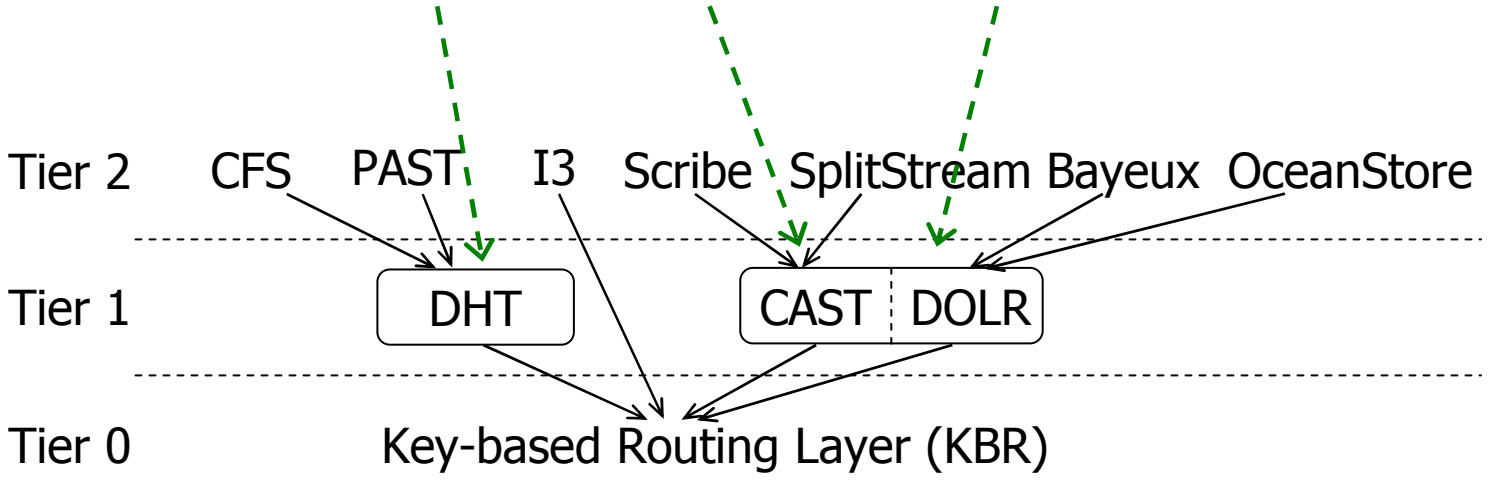
用語

- ルーティング
  - 経路を決めること。そのための諸々の作業。主に、経路表の構築・保守。
- フォワーディング
  - 次ホップにまわしていくこと。

構造化オーバレイの基本

- 肝は **ルーティング/フォワーディング**（のアルゴリズム）
- 資源にかかる負担が、(たいてい) ノード数 n として O(log n)。
  - フォワーディング時のメッセージ数など。
  - cf. Gnutella 等で行われる、非構造化オーバレイ上の flooding
- 様々なアルゴリズムが提案されてきた。
  - CAN, Chord, Tapestry, Pastry, Kademlia, Koorde, Broose, Accordion, ORDI, DKS, D2B, Symphony, Viceroy, …

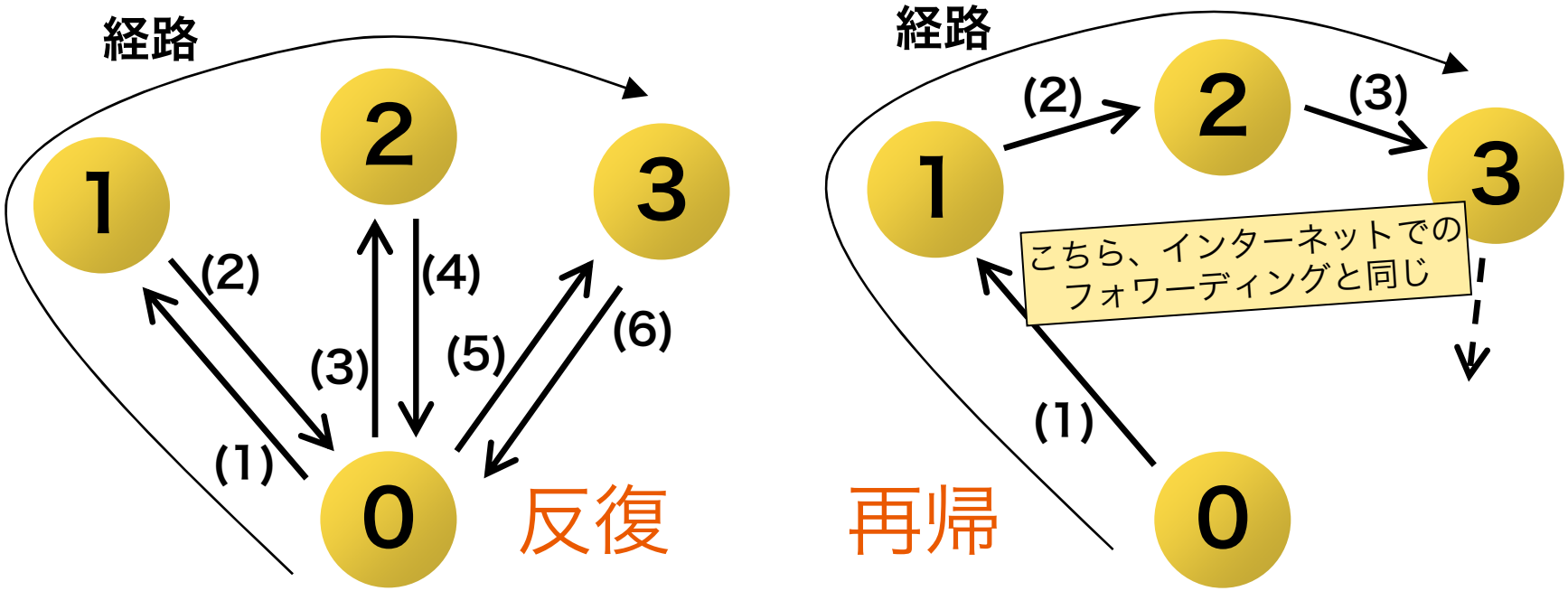
- Structured オーバレイ上に、いろいろなサービスが載る。
  - 分散ハッシュ表 (DHT), マルチキャスト, メッセージ配送, …



Cited from [Dabek03]

フォワーディング様式
forwarding (lookup) styles

- 反復 / 再帰 フォワーディング: iterative / recursive forwarding
  - ルーティングのアルゴリズム (Chord, …) とは、たいてい独立。



- Iterative / recursive routing と呼ばれることが多いが、不正確。
  - routing とは「経路情報の構築」「次ホップの決定」を指す。次ホップへの転送は forwarding。

分散ハッシュ表 (DHT)

- 構造化オーバレイによる、ひとつのサービス。pure P2P な DB。key-value store。
  - put (key, value)
  - get (key)
  - キー・値ペアの削除

- 手順
  - put: keyをキーとしてルーティングを行い、担当ノードにキー・値ペアを保持させる。
  - get: keyをキーとしてルーティングを行い、担当ノードが保持している key に対応する値をもらう。

  - 注) key が ID になっていない場合、ハッシュ値を求めて ID にしておく。例えば（暗号学的ハッシュ関数） SHA1 を通すと、160ビットの ID を得られる。

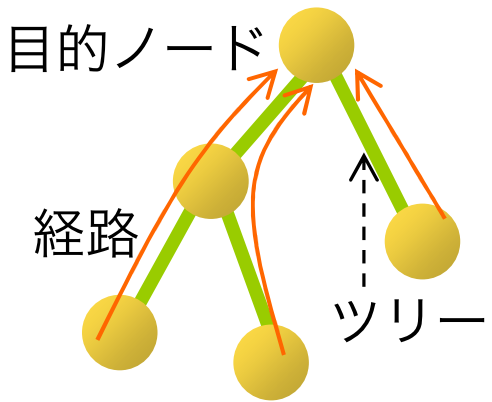
分散ハッシュ表 (DHT)

- 応用: もろもろの名前解決や位置解決
  - ホスト名 → IPアドレス, 名前 → 電話番号, 曲名 → 楽曲ファイルやそのURL などなど。
  - DNS を作ってみました、という研究は多い。
    - “A Comparative Study of the DNS Design with DHT-Based Alternatives”, Proc. INFOCOM 2006.

- 弱点: (そのままでは) 範囲検索ができない。
  - 例: 9月1日〜3日の…, 経度緯度が…
  - 範囲検索が可能な構造化オーバレイ: Skip Graph
  - DHT上で範囲検索、という研究も多い。
    - 中台さん (NEC) は Overlay Weaver上に範囲検索を実装。

構造化オーバレイ上の アプリケーション層 マルチキャスト(ALM)

- ある ID に対して複数のノードからルーティングすると、それら経路の集合はツリーを成す。
  これを配送木として利用する。
- チャネルが ID で表されるので、多チャネル。
  ≠ ブロードキャスト
- 処理
  - あるチャネルにsubscribeするノードは、チャネルの識別子をキーとしてルーティングをする。
  - 経路上のノードは、1ホップ手前と1ホップ先のノードを、それぞれツリーの子と親として記憶する。
  - 各ノードが、ツリー上の親子関係に沿ってトラフィックを転送する。



マルチキャストの応用

- 同報通信を行うもろもろの応用
  - グループチャット・音声 / ビデオ会議
  - 放送
  - VPN
    - L2 のブロードキャストに使える?
    - 例: x-kad: Kademliaというアルゴリズムを応用した VPN
  - 分散処理
    - コードの配布やプロセッサ間同期、ブロードキャスト

- 配送木を使って、エニーキャストも可能。
  - グループ中のノードどれかにメッセージ配信
  - 応用：サーバ群の負荷分散

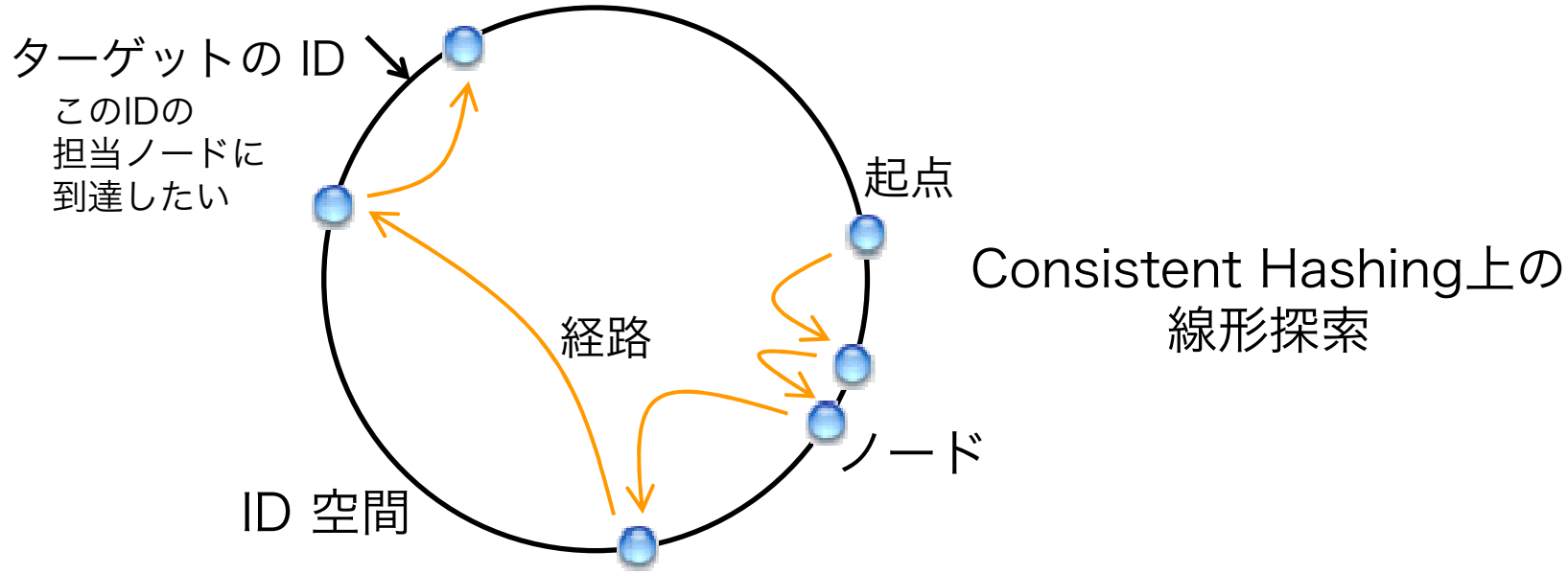
構造化オーバレイの（ルーティング）アルゴリズム

- 宛先 ID に（数値的に）より近い ID を持つノードに、要求をまわしていく。
  - いつかは最も近い ID を持つノードに辿り着く。

- 近づき方の、アルゴリズムごとの違い
  - Chord
    - ID 空間を時計まわりに近づいていく。
  - Pastry, Tapestry
    - ID を、上位桁から順に、b (例 4) ビット単位で揃えていく。
    - Pastry は、最後の 1ホップは違う方法で近づく。
  - Kademlia
    - ID を、上位桁から順に、1 ビット単位で揃えていく。

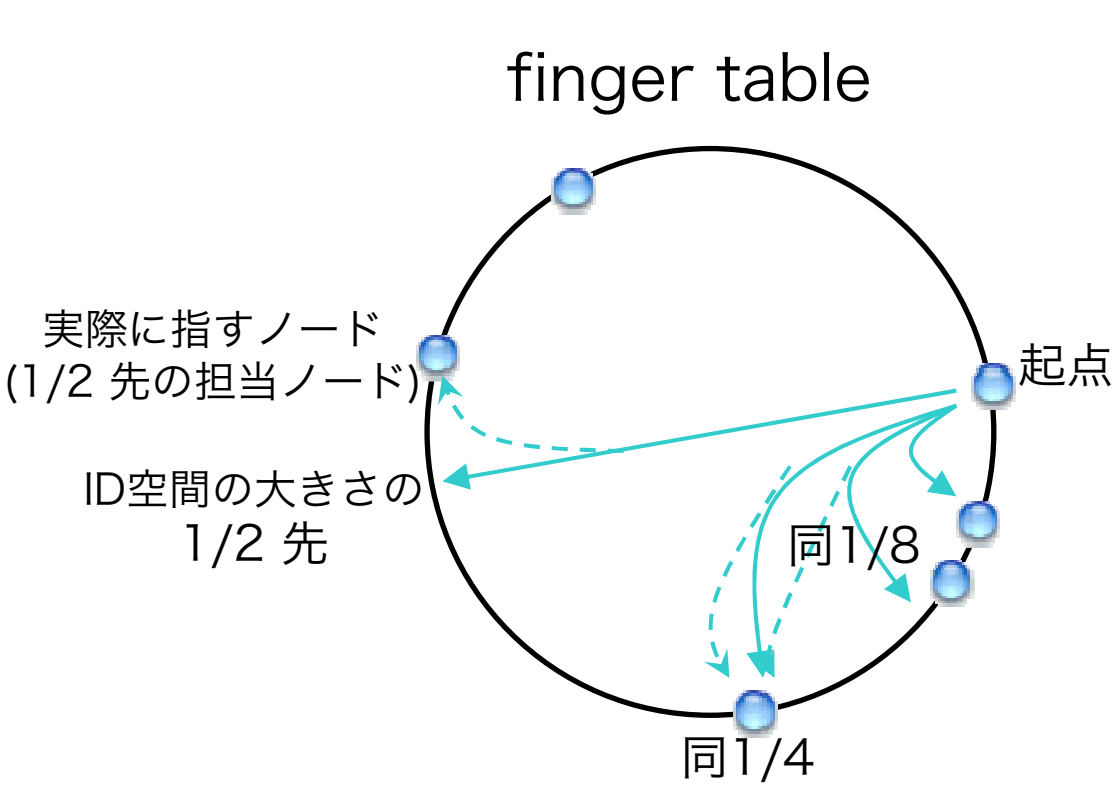
Chord

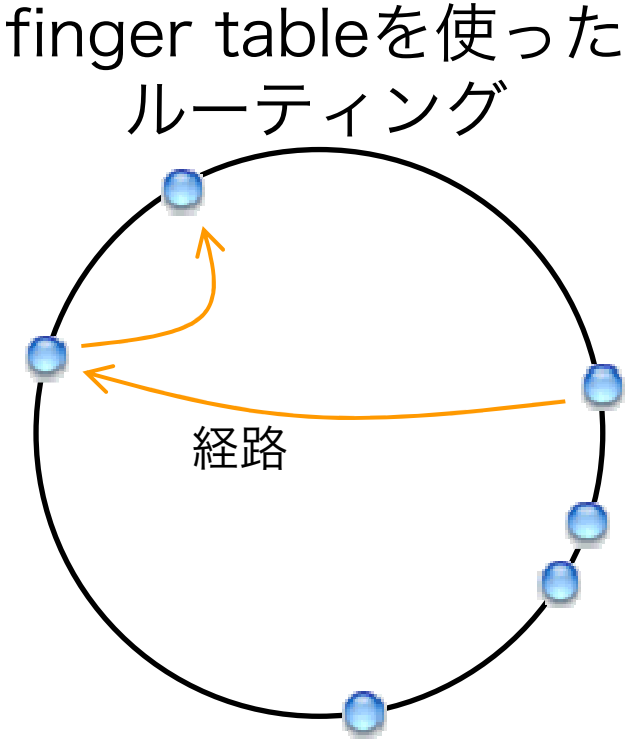
- consistent hashing 上の線形探索 + finger table
  - finger table: ショートカットリンク
- consistent hashing 上の線形探索
  - ID空間を時計まわりに進む。
    - 各ノードは、次のノード (successor) へのリンクを持つ。
  - 下図の場合、5 ホップ
  - オーバレイ上のノード数を n とすると、ホップ数は O(n)。残念！



Chord

- finger table
  - ショートカットのためのリンク
  - 下図のルーティングの場合、2ホップ。
  - ホップ数 O(log n)。万歳！





Pastry, Tapestry

- Plaxtonらの方法 を用いる
  - 上位桁から bビット単位で揃えていく。
- 例
  - b = 4 とする → 16進数の一桁ごとに揃えていく。
  - ルーティングの宛先 ID: 437A (16進数) とする。
    - 本当は、160ビット (Tapestry) や 128ビット (Pastry)。
  - 手順
    - ID が 4XXX であるノードにまわす。
    - ID が 43XX であるノードにまわす。
    - ID が 437X であるノードにまわす。
    - …
  - そういうIDを持つノードが経路表に載っていなければ、次善のノードにまわす。
    - 43XX がなければ、44XX にまわす。

Plaxtonらの方法の経路表

- 例: ID が 10DF であるノードが持つ経路表
  - この場合、サイズは 16 列 x 4 行。
  - n 行目には、n – 1 桁目まで自分と ID が一致するノードが入る。
  - ID とコンタクト先（IPアドレス等）の組が入る。空の場合もある。

	0	*1*		…	*D*	*E*	*F*
1桁目		自身			DXXX	EXXX	FXXX
2桁目	自身				1DXX	1EXX	1FXX
3桁目					自身	10EX	10FX
4桁目		10D1			10DD	10DE	自身

Pastry と Tapestry の違い

- Pastry
  - 前頁の経路表に加えて、ID が数値的に近いノードの集合 leaf set も保持する。
    - 8ノードだったり 32ノードだったり。
    - 次ホップを決める際、まずは leaf set を見る。
- 違い
  - エントリが空だった場合の、次善のノードの決め方。
    - Pastry: とにかく ID が数値的に近いノード。
    - Tapestry: 表を右に見ていく。自身にぶつかったら、下段に降りる。
  - おまけ：広域分散ストレージ OceanStore のルーティングアルゴリズムは、Tapestry + bloom filter

オーバレイへの参加・経路表の保守

- 参加の際は、自身および他ノードの経路表を更新（作成）する。

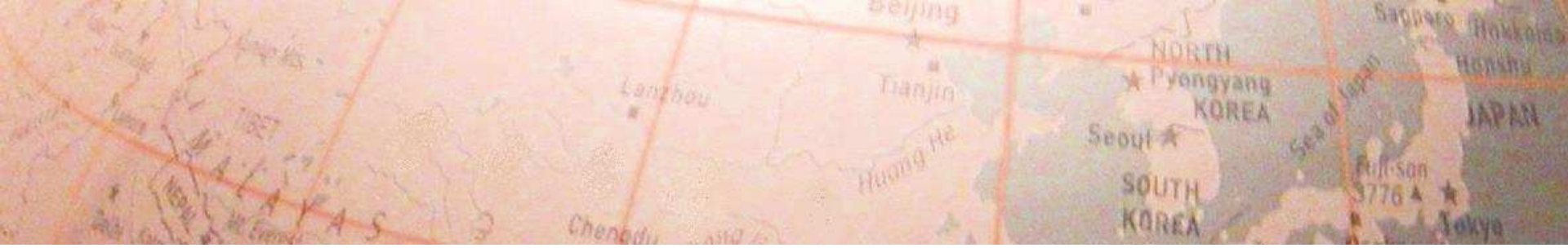
- 新ノードの参加や既存ノードの脱退に応じて、各ノードは経路表を保守する。

  → これこそつまり、ルーティング

- なかなか複雑。
  - アルゴリズムごとの詳細は、ここでは割愛。

オーバレイへの参加・経路表の保守

- 各アルゴリズムが、プロトコル・手順を規定している。
  - 参加の際は、まず、自ノードの ID を宛先としてフォワーディングする。
  - 参加の時点で、関係する全ノードの経路表を更新し切ってしまうか否か：
    - 更新し切る
      - Chord （論文Fig.6のアルゴリズム）, Pastry, Tapestry
    - 保守によって、じょじょに更新される
      - Chord（通常のアルゴリズム）, Kademlia
  - 保守
    - 定期的に通信して行う
      - Chord, Pastry, Tapestry
    - 定期的な保守が不要
      - Kademlia: ルーティング目的の普段の通信を手がかりに保守する。



サーバ側技術と peer-to-peer 由来技術の関係

構造化オーバレイという枠組み

- Wikipedia の DHT についての項
  - degree (次数)　route length (経路長)

O(1)	O(log n)	← constant-degree DHT
O(log n)	O(log n / log log n)	
O(log n)	O(log n)	← 当初の DHT, 構造化オ
O(n^(1/2))	O(1)	

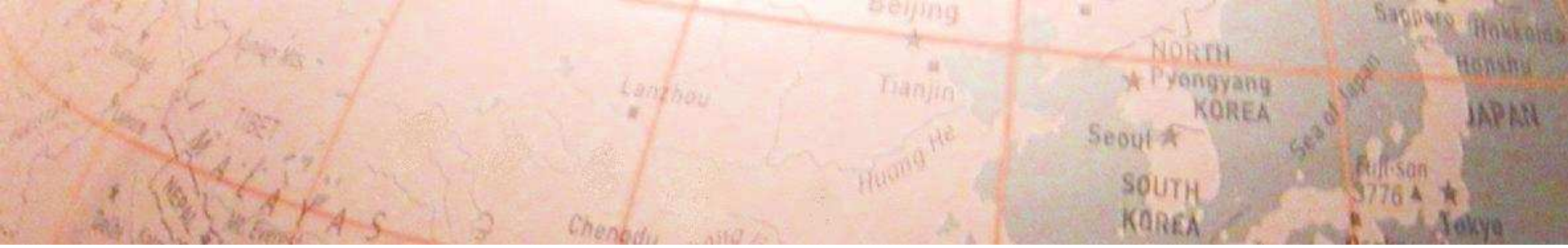
  - no-hop の key-value store は、こう位置づけられる

O(n)	**1**

- 「DHT (分散ハッシュ表)」は、
  従来的には、multi-hop が前提
  → no-hop の key-value store を DHT とは呼ばない。
- しかし no-hop であっても DHT / 構造化オーバレイの枠組み中に位置づけてとらえることが自然。

構造化オーバレイという枠組み

- no-hop … も、
  構造化オーバレイの中に位置づけられる。
  - 構造化オーバレイのルーティング方式
    Chord, Pastry 等も、
    パラメータを調整することで no-hop にできる。
    - 若干の手直しはしたいところだが: メンバ情報の流布等
  - Overlay Weaver [首藤ら2005] も、パラメータ次第で、
    no-hop の key-value store として動かせる。



Azure Services Platform と構造化オーバレイ

Azure 中の構造化オーバレイ

- 構造化オーバレイの使われ方
  - DHT　- **分散ハッシュ表**
    - SQL Data Services の表は、Partition ごとに担当ノードが保持する。注: IDが近いノードに複製は作る。
    - Service Bus (pub/sub によるメッセージ配送サービス) では、エンドポイント (sb://…) の担当ノードが、そのエンドポイントに subscribe したノード一覧を管理する。
      - これ、i3 [Stoica2002] そのまんま。
  - DOLR　- 宛先IDを担当するノードへの**メッセージ配送**
    - Fabric Controller での Messaging。

- Azure のルーティング方式
  - 両方向版 Chord とでも呼ぶべきもの。
    - 時計回り一方向 (Chord) ではなく、両方向に進める。これがいいのかどうかは ???

multi-hop はペイするか？

- cloud のバックエンドは、たいてい no-hop。
  - Dynamo (Amazon), ROMA (楽天技術研究所), …
  - 論文より “Dynamo can be characterized as a zero-hop DHT, where each node maintains enough routing information locally to route a request to the appropriate node directly.”
- no-hop の適応範囲は？
  - ノード数が増えると苦しい。
  - churn (ノードの出入り) が激しくなると苦しい。

  → そこそこ安定した、1,000 ノードくらいの環境？
- cloud の裏側はどうか？

multi-hop はペイするか？

- 1つのサービスが 1,000台を要することは、どのくらいあるか？

- 「Multi Tenant」 (by Salesforce.com) が鍵 ???
  - 異なるサービスの間で多くの機材、分散システムの系 (含 オーバレイ) を共用、融通する。
    → ユーティリティとしての cloud

multi-hop の オーバヘッドを減らす方策

- 一度フォワーディングを済ませたら、
  経路をショートカットして、
  ターゲット / 宛先 ノードと直結してしまう。
  - Service Bus の「Direct Connections」は、これ。
- no-hop 用の経路表 (全ノード一覧) も持つ。保守する。
  - あまり賢くない。
- 構造化オーバレイのパラメータを調整して、
  no-hop で直接到達できる割合を高くする。
  - 要 研究
  - overlays for clouds …首藤研のネタのひとつ？ ☺