下位アルゴリズム中立な DHT 実装への耐 churn 手法の実装

首藤 一幸

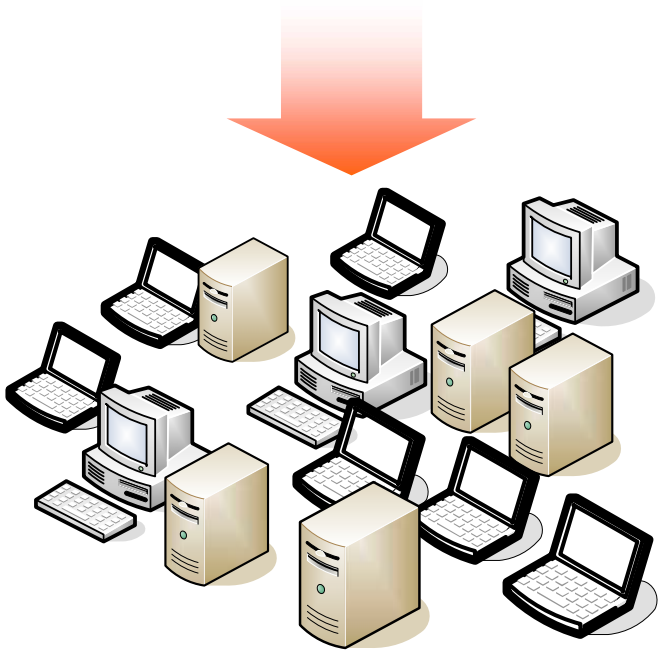
ウタゴエ (株)

UTA GOE
UNIVERSAL GRID

分散ハッシュ表 (DHT)

- pure P2P なハッシュ表
  - 対等なノード群が、中心となるノードなしに、非集中的・自律的に達成する。
- 応用の範囲が広い。
  - 様々な名前解決に用いることができる。
  - cf. Berkeley DB
  - 例: DNS
    “A Comparative Study of the DNS Design with DHT-Based Alternatives”, Proc. INFOCOM 2006.
  - 例 (not DHT): Amazon の Dynamo
    “Dynamo: Amazon’s Highly Available Key-value Store”, Proc. SOSP 2007.

put (key, value)
get (key)
remove (key, …)



churn

- churn: 絶え間ないノードの出入り
  the continuous process of node arrival and departure

- 非集中分散 (peer-to-peer) システムには
  churn 耐性が求められる。
  - 出したいメリット:
    - 低い管理・提供コスト
    - 高いスケーラビリティ
    - 並以上 ～ 高い信頼性
  - アプローチ
    - ×　ノード個々の信頼性を高める → 高コスト
    - ○　ノード個々の信頼性・可用性はあまり求めない。churn 前提。
      離脱や加入 (故障や復帰) が起こることを前提とする。

研究の概要

- DHT を対象とした
  耐 churn 手法とそれらの効果を示す。

- 特徴
  - どの手法も、DHTの下位の
    ルーティングアルゴリズムに依存しない。
    → 様々なアルゴリズムと組み合わせられる。
    - アルゴリズム: Chord, Kademlia, Pastry, Tapestry, Koorde, …
      それぞれに特性がある。

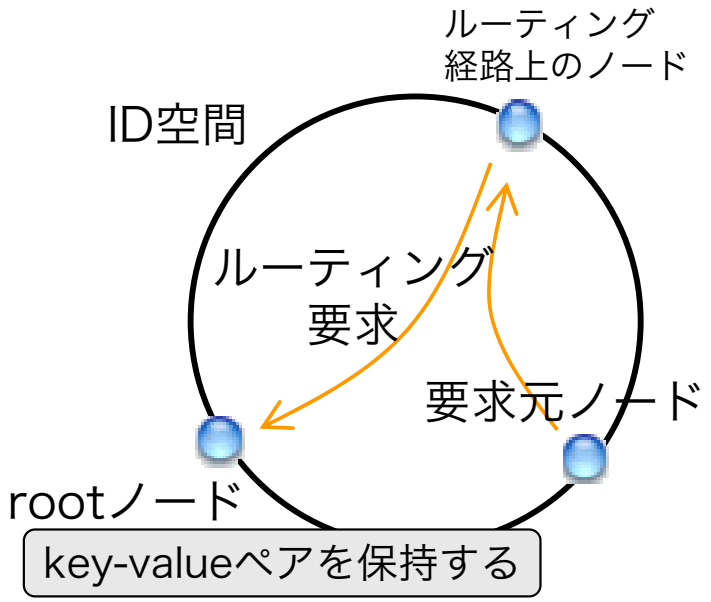
- 貢献
  - ルーティングアルゴリズムに対して中立な
    耐 churn 手法の有効性を初めて実証した。
    - 手段: Overlay Weaver に実装し、get 成功率を計測した。

churn 問題

UTA GOE
UNIVERSAL GRID

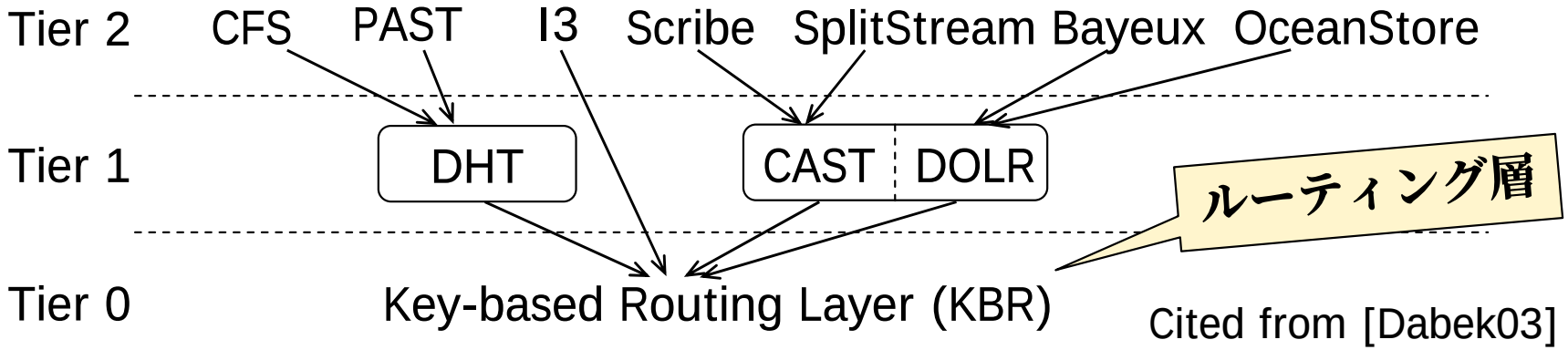
DHT: 構造化オーバレイの一応用

- DHT は構造化 (structured) オーバレイの一応用であるという整理に従うと
  - **put**: key を宛先としてルーティングを行い、到達した root ノードに key-valueペアを保持させる。
  - **get**: key を宛先としてルーティングを行い、到達した root ノードから値を取得する。



注: アルゴリズムによっては、
ID空間を円環としてとらえることが
適切とは限らない。

Dabek らによる 構造化オーバレイの抽象化



Cited from [Dabek03]

churn 状況下では

- put したはずの値を get できない
  ことがある → get 失敗

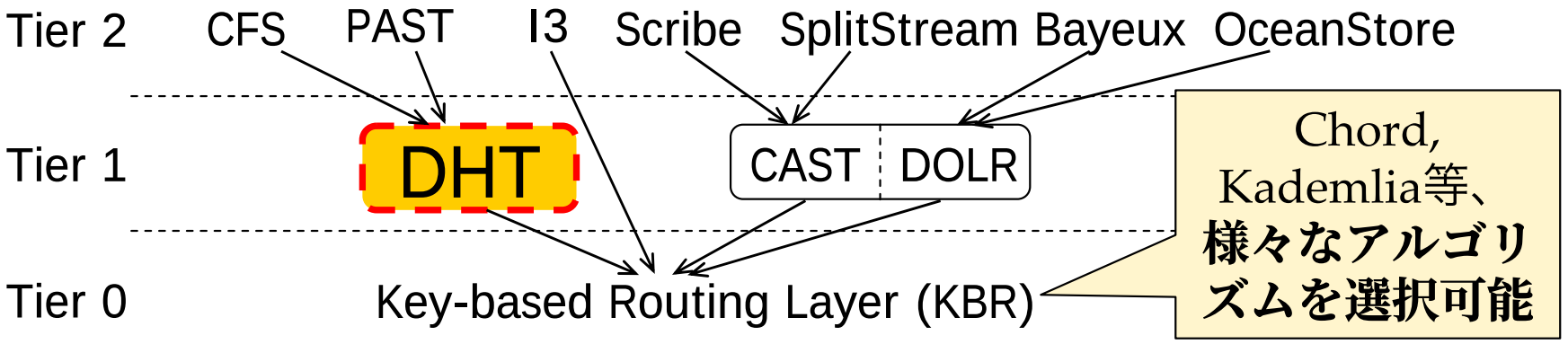
- 原因
  - put 後に (ノード離脱によって) key-value ペアが消滅した。
  - root ノードが key-value ペアを保持していない。
    - put 後に加入したノードが新たな root になった。
    - put 時のルーティングが root に到達していなかった。
      (経路表が不完全、等)
  - ルーティング要求を転送中のノードが離脱した。
    (recursive ルーティングでのみ発生)

耐 churn 手法

UTA GOE
UNIVERSAL GRID

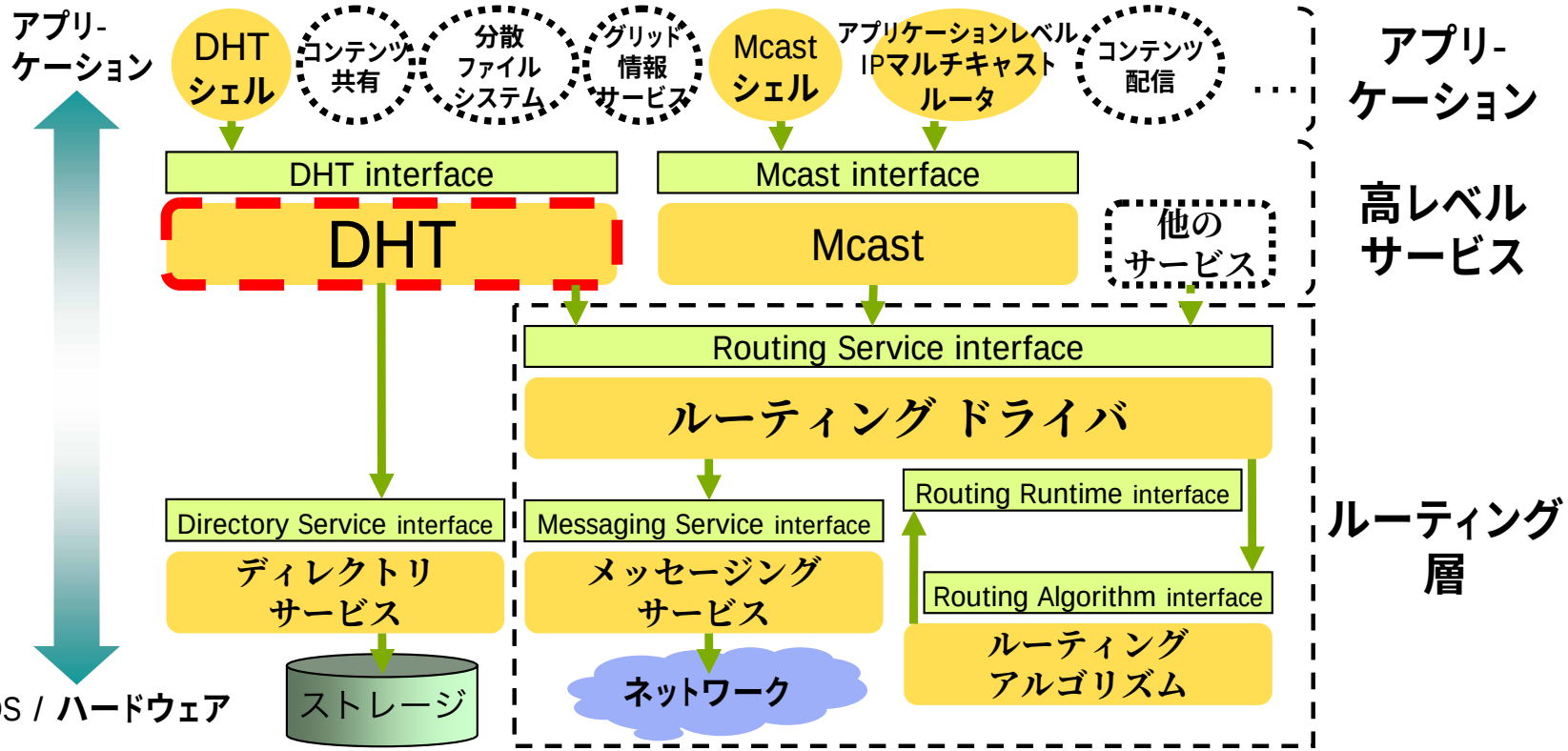
耐 churn 手法の実装と効果測定

- DHT 層を対象とした 4つの手法を実装。
  - 複製
  - 加入時委譲
  - 複数 get
  - 自動再 put
  - **特徴**: DHT 層への実装であるため、様々なルーティングアルゴリズムと組み合わせられる。
- 複数のルーティングアルゴリズムに対して効果を測定。
  - 貢献: アルゴリズム中立な耐 churn 手法の有効性を実証。

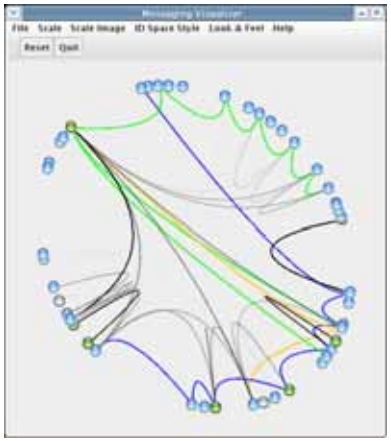


Overlay Weaver の DHT 層に実装

- Overlay Weaver [首藤06, Shudo07]
  - 構造化オーバレイのライブラリ・研究プラットフォーム
  - Dabek らの抽象化に従っている。
  - ルーティングアルゴリズムを差し替え可 & 容易に実装可能。
    - Chord, Kademlia, Pastry, Tapestry, Koorde の実装を含む
  - 実環境での動作 & PC 1台で 8,000ノードをエミュレート



可視化ツール

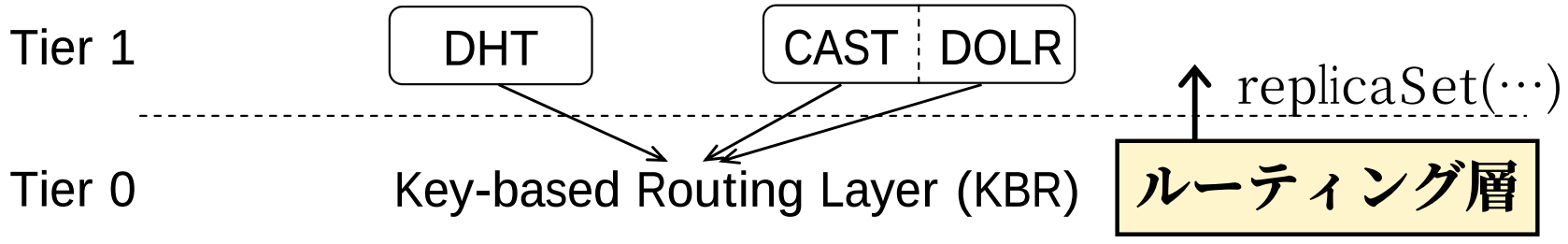


準備

- ルーティング層が上位層に対して、
  ルーティングの結果として、
  root ノード候補の**リスト**を返すようにした。
  - root として適切な順にノードを並べたリスト
  - 自動再 put 以外の 3手法が、この機能を利用。

- Dabek らの抽象化でも、ルーティング層が replicaSet(key, max_rank) を提供。ただし
  - 名が表す通り、複製だけを考慮。
  - API の提供が論文の主旨であり、実証は伴っていない。
  - 機能も若干異なる: ローカルに呼ばれる手続き

Dabek らによる 構造化オーバレイの抽象化



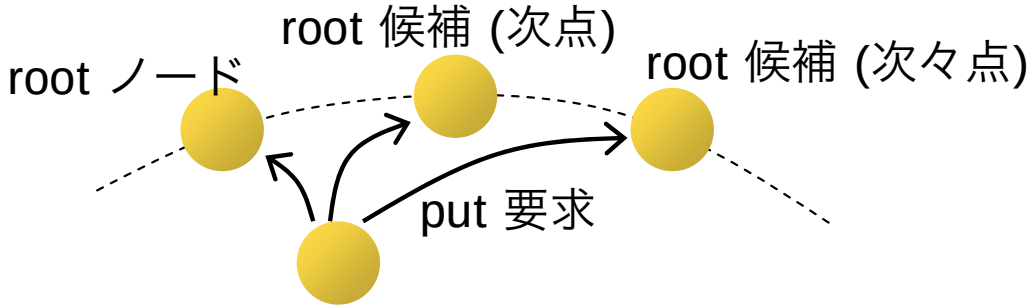
実装した耐 churn 手法

- 複製
  - put時、root ノード以外にも key-value ペアを保持させる。
- 加入時委譲
  - オーバレイへの加入時、周辺のノードから key-value ペアの委譲を受ける。
- 複数 get
  - get 時、root ノード以外にもいくつかのノードに問い合わせる。
- 自動再 put
  - 保持している key-value ペアを、自動的に DHT 上に put しなおす。

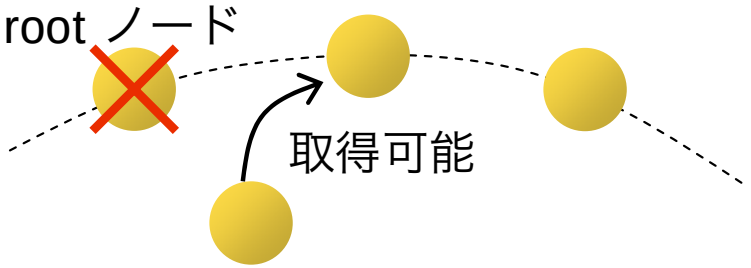
複製

- put 時、root 候補の数ノードに key-value ペアを保持させる。
  - 効果
    - put 後に root ノードが離脱した場合でも、値を取得できる。
  - パラメータ
    - 複製数
    - 複製の put 要求を行うノード: put 要求元 or root ?

put 時



root ノード離脱後

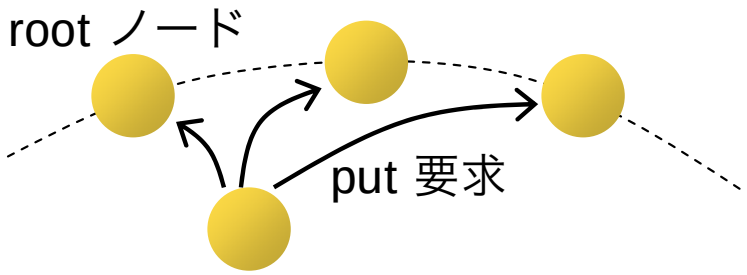


複製手法についての考察

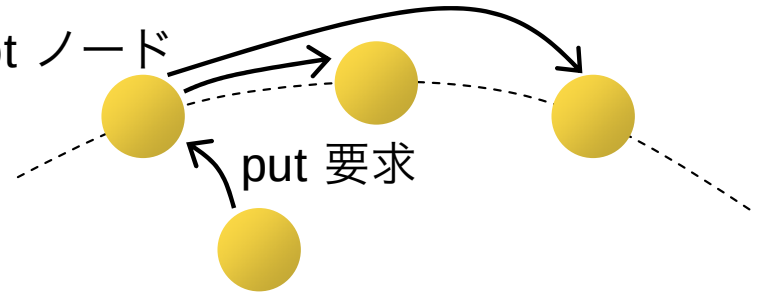
- key からいくつかのキーを導出して、それらキーに対して put する、という方法も考えられる。
  - 例: ① key ② key xor 010.. ③ key xor 100.. ④ key xor 110..
  - そのぶん、ルーティングの回数、ひいてはトラフィックが増してしまう。→ 今回は不採用

- 複製の put 要求を、put 要求元と root ノードのどちらが行うか？ どちらも選択可。
  - どちらが行っても通信の回数は同一。
  - ノードの ID とネットワーク近接性に関係がある場合 (PIS: Proximity Identifier Selection) には、効率に差。
    → 今回は影響なし。

put 要求元ノードが行う

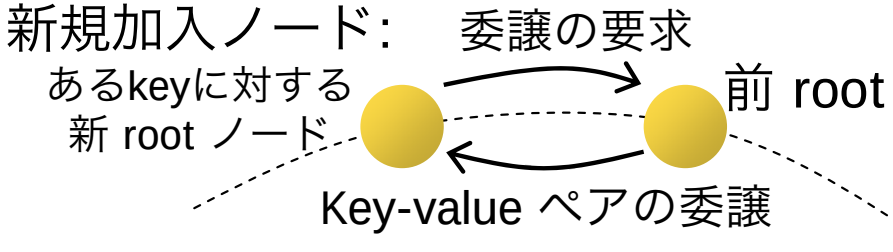


root ノードが行う



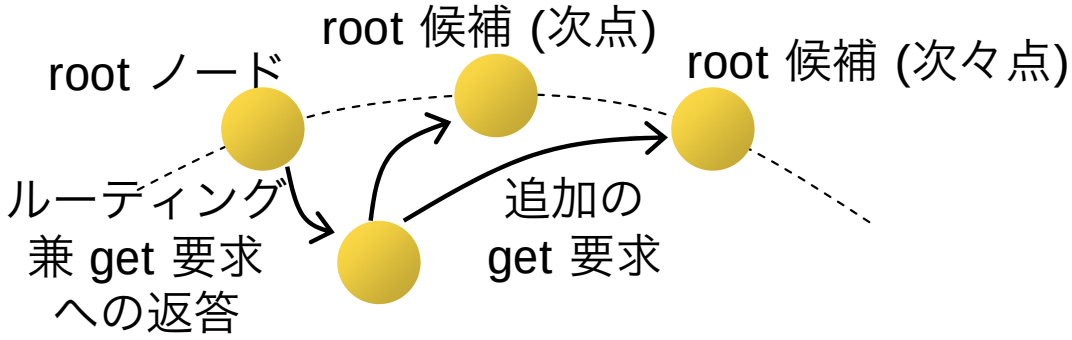
加入時委譲

- DHT に加入してきたノードに対して、そのノードが root となるような key-valueペアを他のノードから委譲する。
  - 方法
    - 加入時ルーティングで得られる root 候補いくつかに、委譲を依頼する。依頼を受けたノードは、加入ノードの方が root として適当な key-value ペアを返す。
  - 効果
    - put 後に加入したノードが root となるような get 要求に応えられる。
  - パラメータ
    - 何ノードに対して委譲を依頼するか。



複数 get

- get 時、root 候補の数ノードに問い合わせる。
  - 効果
    - put 後に加入したノードが root となるような get 要求に応えられる場合がある。
      - 加入前に root だったノードに対しても問い合わせる可能性があるため。
      - 加入時委譲と効果が重複。
    - put時のルーティングが root ではなく次善のノードに達してしまった場合にも、有効。
      - 経路表が不完全な場合に起こり得る。
  - パラメータ
    - 要求先ノード数



自動再 put

- 各ノードが、保持している key-value ペアを、たまに DHT 上に put し直す。
  - 通常の put と同じく複製も作る。
  - 今回の４手法のうち、唯一、root ノード候補のリストを (直接は) 使わない。
  - 効果
    - ノードの離脱に従って減っていく複製を補充する。
    - 加入時委譲と同じ効果もある：適切な root に対する put。しかし、効果は限定的。
      - 自動再 put 処理には時間間隔がある → 加入直後の put には役立たない。
      - 複製数 1の場合、複製補充の効果はない。
        しかし、加入時委譲と同じ効果はある。
  - パラメータ
    - 時間間隔
      - 複数ノードの再 put 処理が同期してしまうことのないよう、乱数で増減させる。

各手法が対象とする get 失敗原因

- **複製**: ノード離脱に伴う key-value ペアの消滅を防ぐ。
- **自動再 put**: 複製数の減少を補う。
- **加入時委譲, 複数 get**: root が保持すべき key-value ペアを保持していない状況への対策。

原因 ＼ 手法	複製	加入時委譲	複数 get	自動再 put
key-value ペア消滅	√			√ (複製が前提)
rootがkey-valueペアを保持せず:				
新ノードが root に		√	√	√ (効果は限定的)
put 時 root 不達			√	√

効果

各手法が get 成功率に与える影響

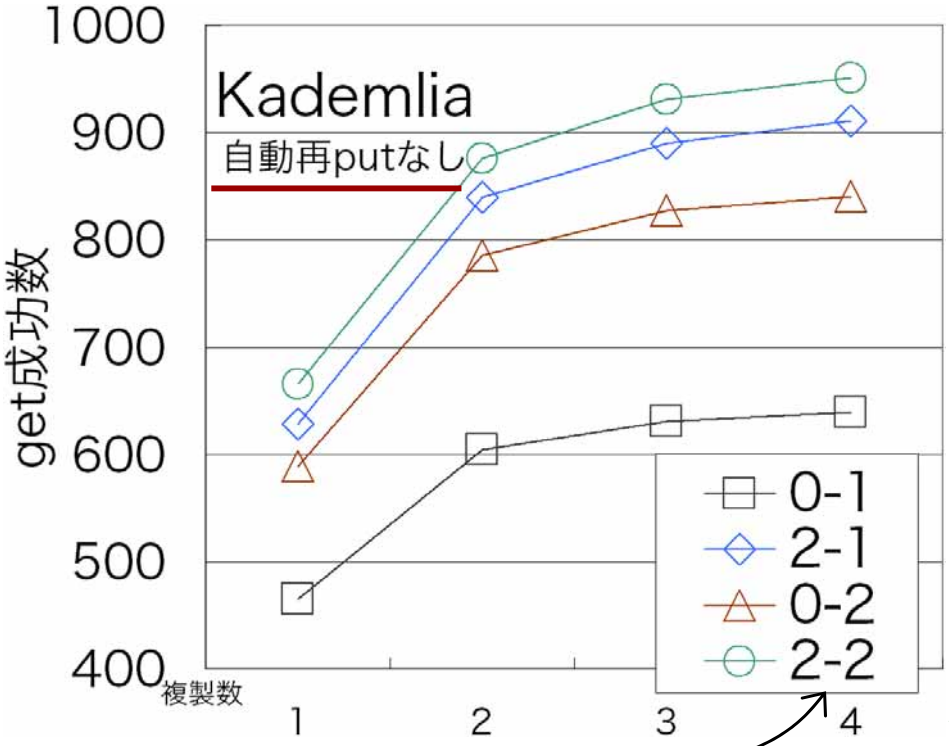
UTA GOE
UNIVERSAL GRID

各手法の効果を計測

- PC 1台の上で
  - 1,000 ノードを動作させ
  - churn を起こし
  - get が成功した回数を数えた。
- 対象
  - Overlay Weaver がサポートする 5つのルーティングアルゴリズムと iterative / recursive ルーティングの全組み合わせ。
- 実験シナリオ
  - 1,000 ノードを起動
  - 全ノードを 0.15 秒毎に DHT に加入させる
  - 1,000 通りの key-value ペアを 0.2 秒毎に put
  - 1,000 通りの key で、0.2 秒毎に get
- 条件
  - put, get を行うノードはシナリオ生成時に乱数で決定。
  - churn は、put 開始時から get 終了までの間、起こした。
  - churn のモデル: ノード離脱の直後に別のノードを加入させる → ノード数は一定。
  - churn 頻度は 2回 / 秒 → 平均生存時間 500秒。タイミングはポアソン分布。
  - 通信のタイムアウトは 3秒、ルーティング全体のタイムアウトは 10秒。
  - 自動再 put の間隔は平均 30秒。
- 実験環境
  - Overlay Weaver 0.6.4 の分散環境エミュレータ
  - x86 用 Java SE 5.0 Update 12
  - x86-64 用 Linux 2.6.21
  - 2.8 GHz Pentium D

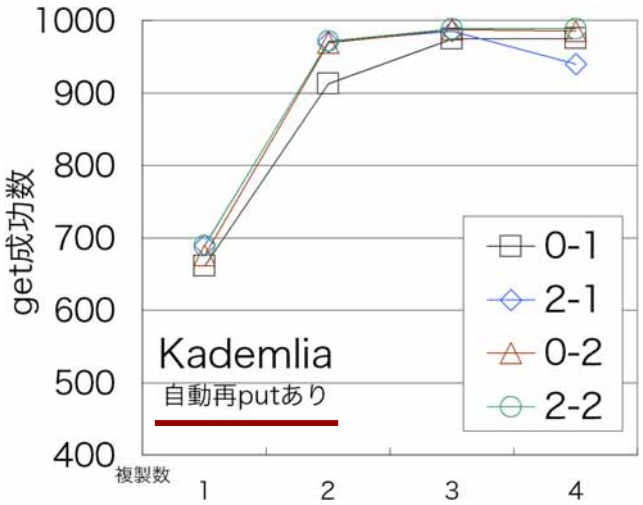
結果と考察： どの組み合わせでも同じ傾向

- 直感的 (当たり前) な結果
  - 複製数が多いほど、get 成功率が上がった。
  - 加入時委譲と複数 get は、どちらも、行った方が get 成功率が上がった。
- その他
  - Pastry と Kademlia では、複数 get (get要求先:2) よりも加入時委譲 (問い合わせ先:2) の方が効果が高かった。Chord ではその逆。
  - 複製数 3 よりも 4 とした方が結果が悪い場合があった。



<加入時委譲> - <複数 get>

- 自動再 put を行うと、加入時委譲や複数 get による差がほとんど見られなくなった。



ルーティングアルゴリズム中立な耐 churn 手法の有効性を実証できた

結果と考察 (cont'd)

- get 失敗の原因
  - 耐 churn 手法が追いつかなかった。
    - 複製すべての消滅, 新規 root への再 put より早い get, など。
  - 加入したノードの経路表が不完全。
    - すると、適切な root に到達しない。
  - 通信のタイムアウトが複数回起き、そのうち、ルーティング自体がタイムアウトした。

- 注: 今回の結果は、アルゴリズム間の優劣を表すものではない。
  - 各アルゴリズムにそれぞれパラメータがあり、中には churn 耐性に影響を与えるものがある。
    - 例: Chord の stabilize 間隔
    - 今回は、Overlay Weaver 0.6.4 の既定のパラメータを採用した。

効果があることはわかった

実地に応用することを考えた場合

- 各手法の採否
- 各種パラメータ

はどう決めたらいい？

UTA GOE
UNIVERSAL GRID

費用対効果

Future work に向けた考察

UTA GOE
UNIVERSAL GRID

費用対効果

- **効果**
  - get 成功率として表れるので、その関数。
- **費用**：耐 churn 手法の適用で増えるもの
  - 通信の回数と量
  - 加入, put, get に要する時間
    - 加入時委譲や複数 get には、余分な時間がかかる。
  - メモリやストレージの消費量
    - 複製数 n の場合、n 倍消費。
    - 組み込み機器ではかなり高価な費用となり得る。
  - プロセッサの処理
    - 応答性, 消費電力, …

従来研究が
着目してきた費用

費用を考える際、
応用の**環境**を想定することが欠かせない

応用ごとの系の振る舞い

- 手法によって、費用が発生するタイミングが異なる。
  - 例えば

手法＼タイミング	加入	put	get	平常時
複製		√		
加入時委譲	√			
複数 get			√	
自動再 put				√

    - 離脱・加入があまり起きないなら、加入時委譲は費用が少ない。
    - 自動再 put は、key-value ペアがある限り、put, get 要求がなくとも費用がかかり続ける。
- 応用によって、各処理の頻度が異なる。
  - DNS: put より get が圧倒的に多い → 複数 get はコスト高

応用ごとに異なる 系の振る舞いを考慮する必要がある

Future work

- 応用と環境を想定して、
  費用 (対 効果) を計測・算出する。

- 応用
  - DNS:　put の頻度 << get の頻度
  - センサネットワーク:　　put ??? get

- 環境
  - ストレージと通信のコスト費

関連研究

- Dabek らの階層モデル [Dabek03]
  - DHT は構造化オーバレイの一応用、という整理
  - 複製機能を実装するために、ルーティング層が提供する API: replicaSet(key, max_rank)
    - 複製だけを想定
    - 実証は伴わず

- Bamboo の耐 churn 手法 [Rhea04]
  - 離脱ノードの検出方法, 通信タイムアウトの調整, proximity neighbor selection の効果を計測
  - ルーティング層に実装する手法である。今回述べた手法とは独立しており、組み合わせることも可能。
  - DHT 実装 Bamboo や、Pastry ベースのルーティング手法が前提。

- ネットワークエミュレータ peeremu を用いた DHT 実装の評価 [加藤06, Kato07]
  - DHT 実装 Bamboo, Chord, Accordion, FreePastry を評価。十数台の　PC で 1,000 ノード規模の実験。churn 時の get 成功率、get の所要時間を計測。

- スーパーノード
  - スーパーノードとして選出されたノードだけでオーバレイ (DHT) を構成。他のノードはスーパーノードを通してオーバレイの機能を利用。
  - オーバレイ (DHT) に要求される churn 耐性を下げることができる。
  - とはいえ、ノードの離脱・加入を想定しないわけにはいかない。

まとめ

- DHT を対象としたいくつかの
  耐 churn 手法と効果を示した。
  - 下位のルーティングアルゴリズムに中立

- ルーティングアルゴリズム中立な
  耐 churn 手法の有効性を実証した。

- 各手法の採否やパラメータ決めに必要となる費用対効果の算出方法を考察した。
  - 要考慮:
    - 応用の環境 (PC, 組み込み, インターネット, 無線, …)
    - 応用ごとの系の振る舞い

参考文献

- [Dabek03] F. Dabek et al., “Towards a Common API for Structured Peer-to-Peer Overlays”, 2003.
- [首藤06] 首藤一幸 et al., “オーバレイ構築ツールキット Overlay Weaver”, 2006.
- [Shudo07] K. Shudo et al., “Overlay Weaver: An Overlay Construction Toolkit”, 2007.
- [Rhea04] S. Rhea et al., “Handling Churn in a DHT”, 2004.
- [加藤06] 加藤大志 et al., “ネットワークエミュレータによる大規模 DHT 性能評価手法の提案”, 2006.
- [Kato07] D. Kato et al., “Evaluating DHT Implementations in Complex Environments by Network Emulator”, 2007.