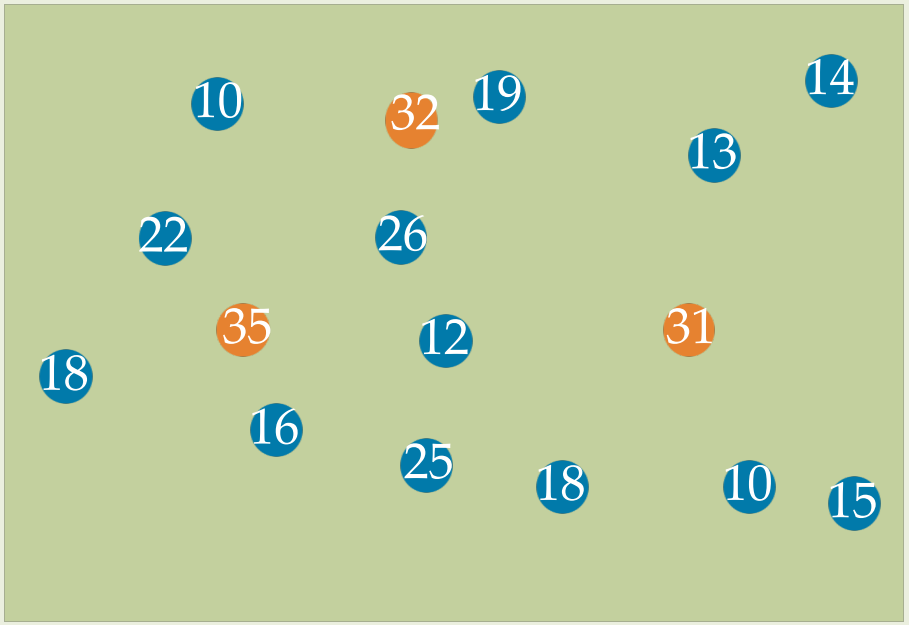
構造化オーバーレイネットワークを用いた 条件付きマルチキャストの 提案と評価

大阪市立大学 大学院工学研究科
安倍広多

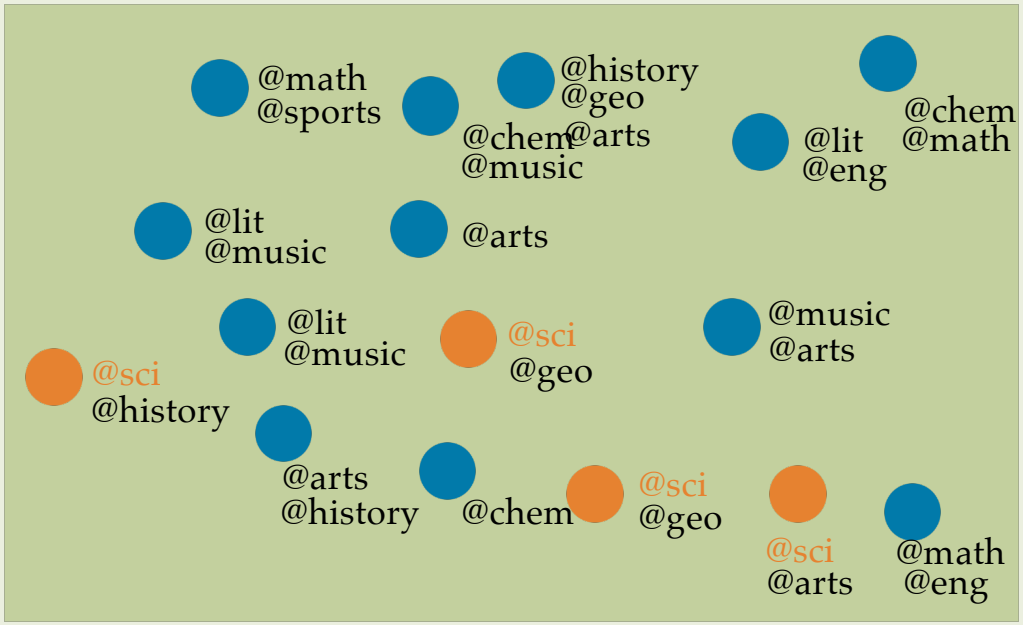
DPSWS 2019

背景 | 条件付きマルチキャストとは

- ネットワーク内の各ノードが何らかの属性値を持つ
- 属性値が指定した条件を満たすノードにマルチキャスト



(例1) 30°C以上のノードにマルチキャスト



(例2) @sciの購読ノードにマルチキャスト (Pub/Sub)

本研究	条件付きマルチキャストを構造化オーバーレイネットワークで実現

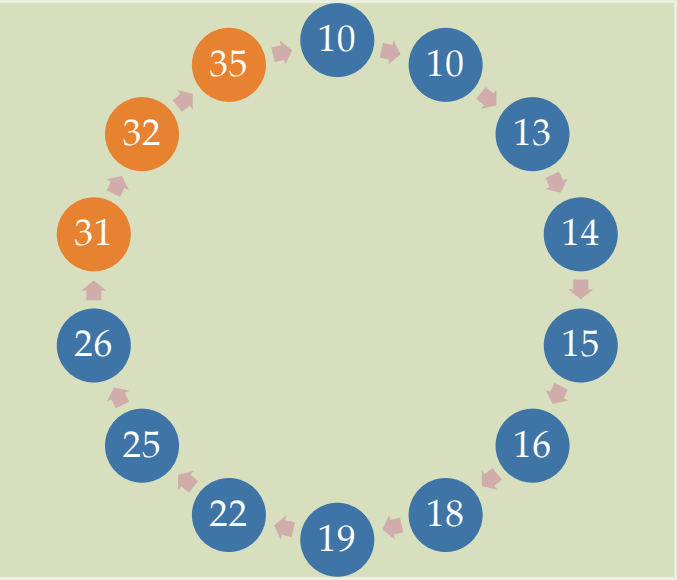
背景 | 従来手法

- 範囲検索が可能な構造化オーバーレイネットワークを使う (Skip Graph, Chord#, Suzakuなど)
- 属性値をキーにマップし，属性値の範囲を指定したマルチキャストを行う
- 欠点

  連続した1次元範囲がマルチキャスト対象となる場合にしか使えない

  属性値が変化したら挿入し直し

  など…

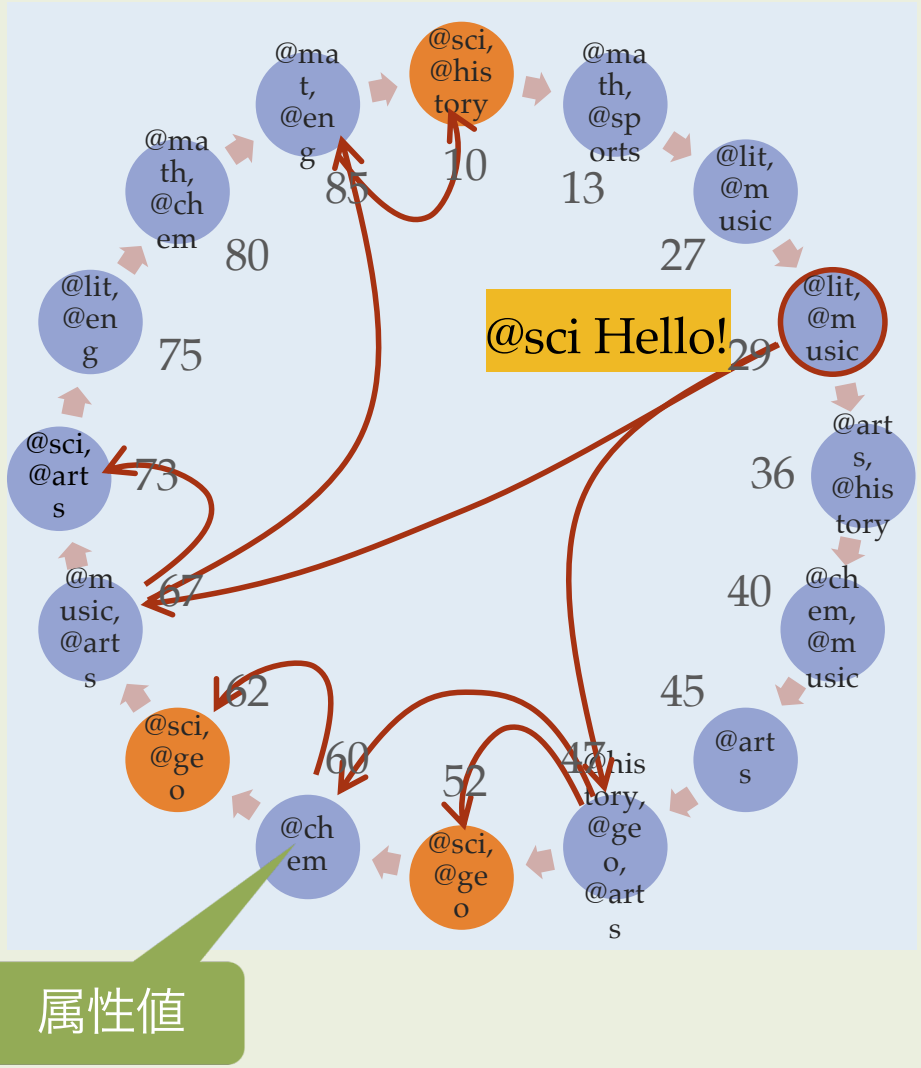


提案手法

提案手法の概要

- 構造化オーバーレイネットワークでリングを形成
- 各ノードはキーと属性値を別々に持つ
- マルチキャストの対象ノードの指定は柔軟
- 属性値が変化しても構造は変化しない
- キーの範囲でマルチキャスト対象を限定できる

  地理的範囲で絞る, etc.
- O(log n)ホップで到達

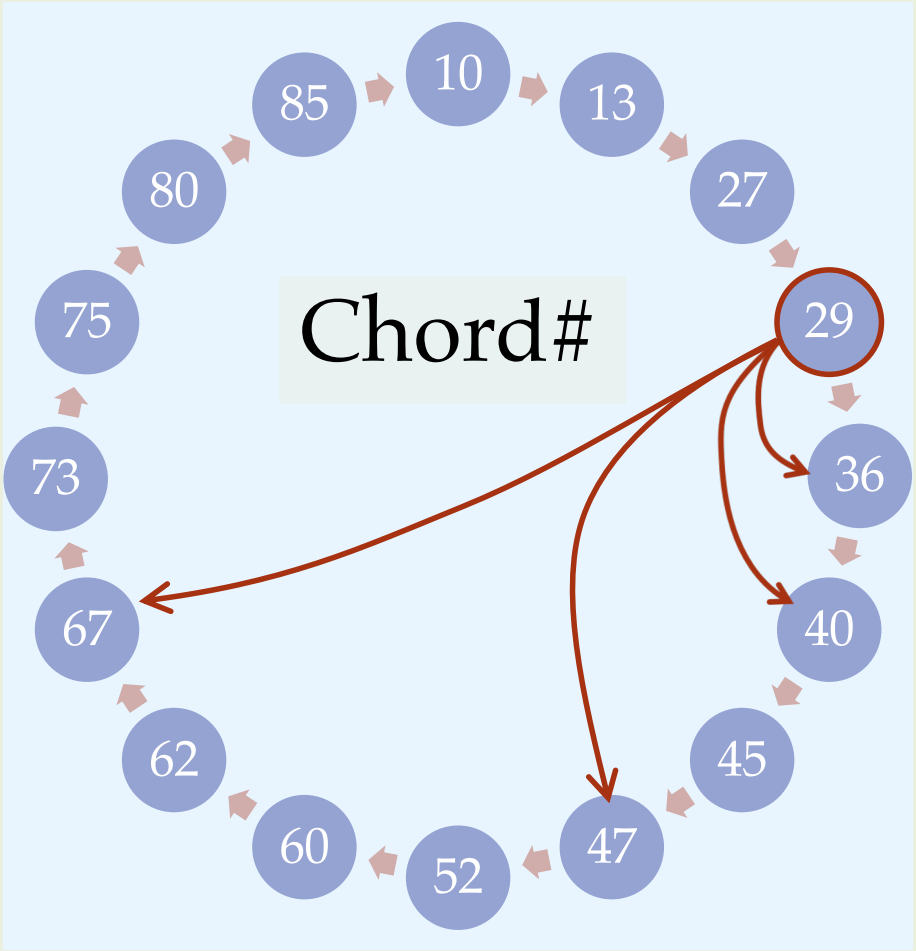


構造 (Chord#)

- 構造化オーバーレイネットワークのベースはChord#
- キー順に並んだリング構造
- 各ノードは2の累乗個離れたノードにショートカットリンクを持つ

29の経路表

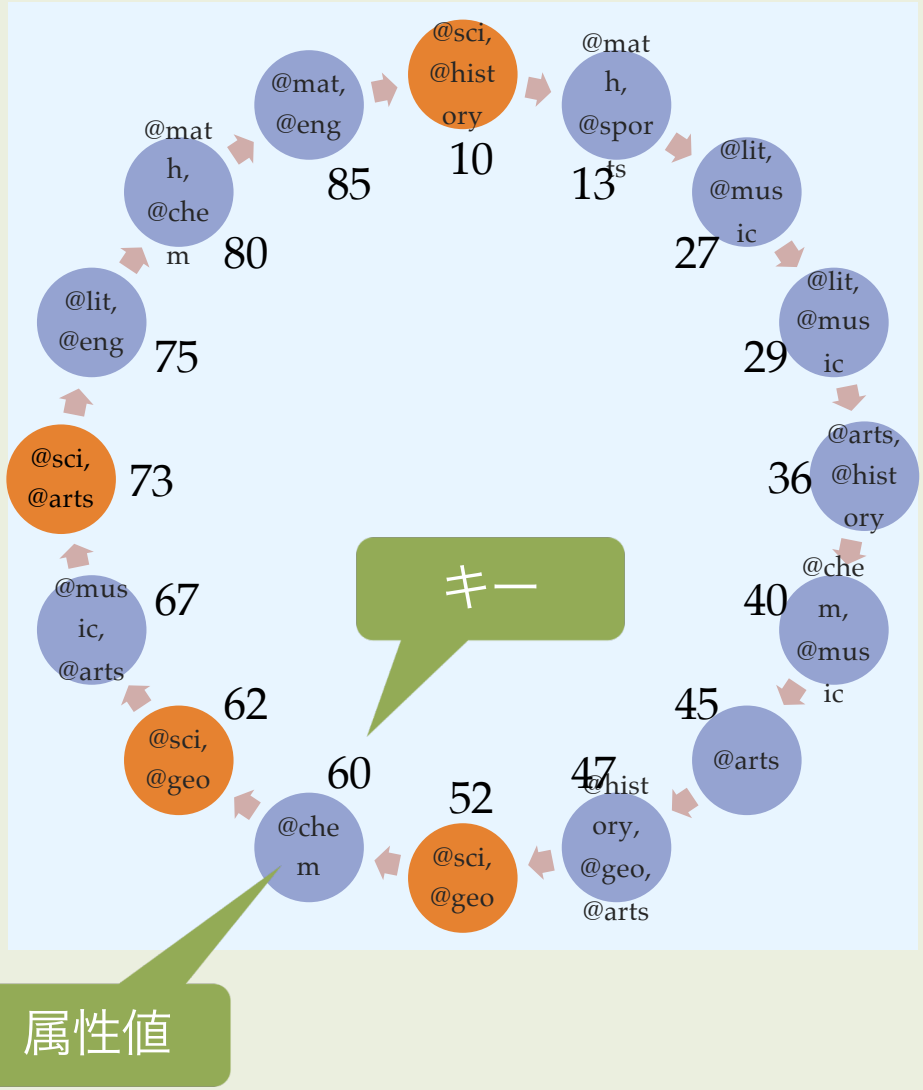
レベル	ポインタ	キー
0	+1離れたノード	36
1	+2離れたノード	40
2	+4離れたノード	47
3	+8離れたノード	67



属性値

- 各ノードはキーに加えて属性値を持つ
- 属性値の型は任意

  例: [“@lit”, “@music”]

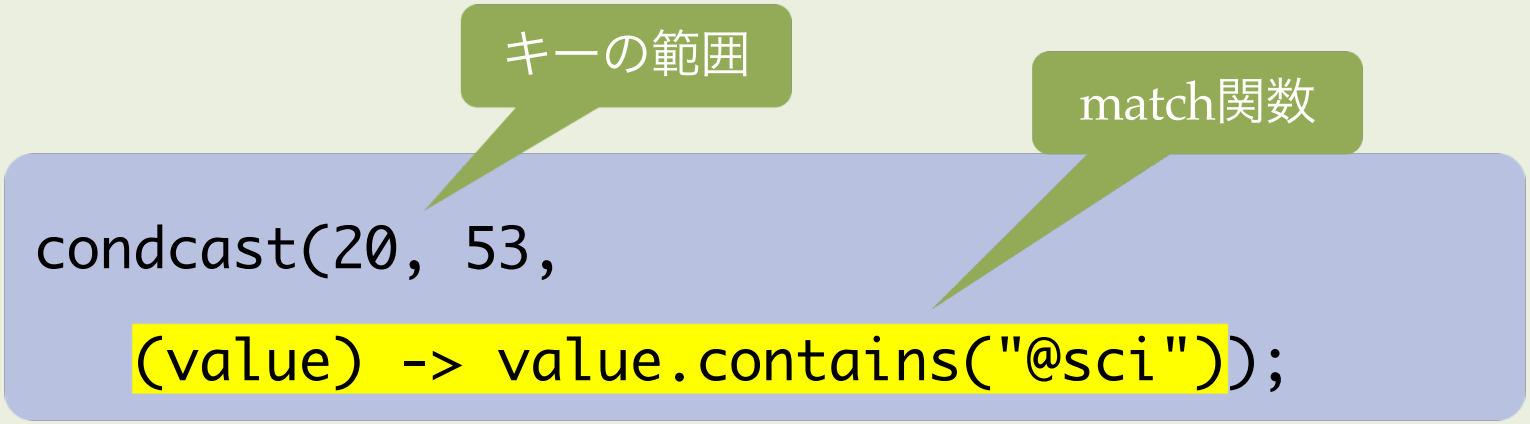


宛先の指定方法

- 宛先ノードの条件はキーの範囲とユーザ定義関数(match)で指定

ノードuが宛先に含まれる ↔ match(uの属性値) = true

- 例

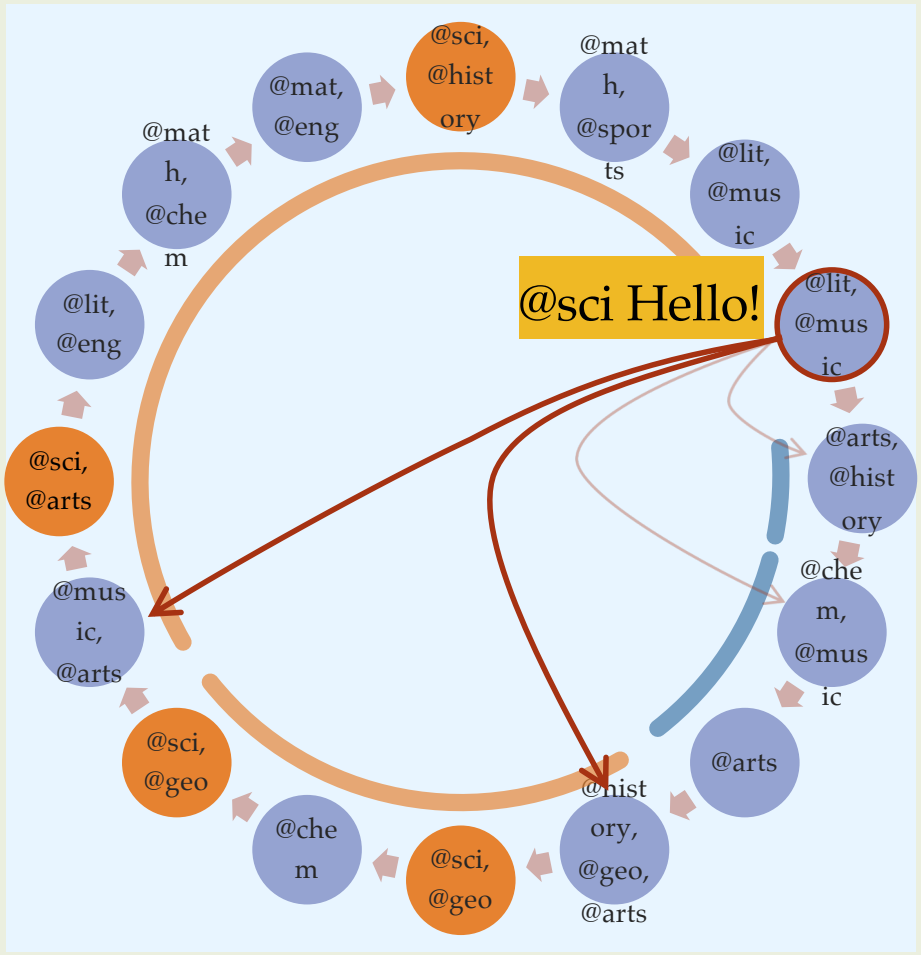


- matchの定義を変えることで柔軟な宛先指定が可能

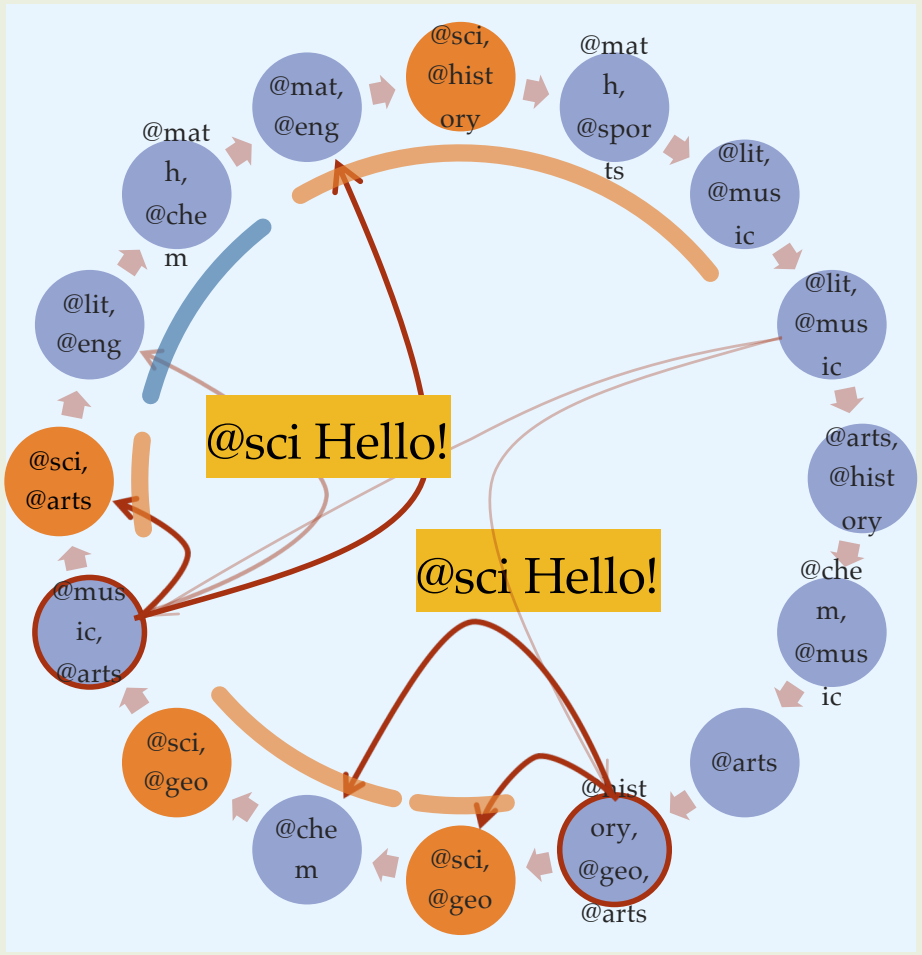
条件付きマルチキャストの実行

- ショートカットの担当区間内に条件を満たすノードがあればメッセージを再帰的に転送

1ホップ目



2ホップ目

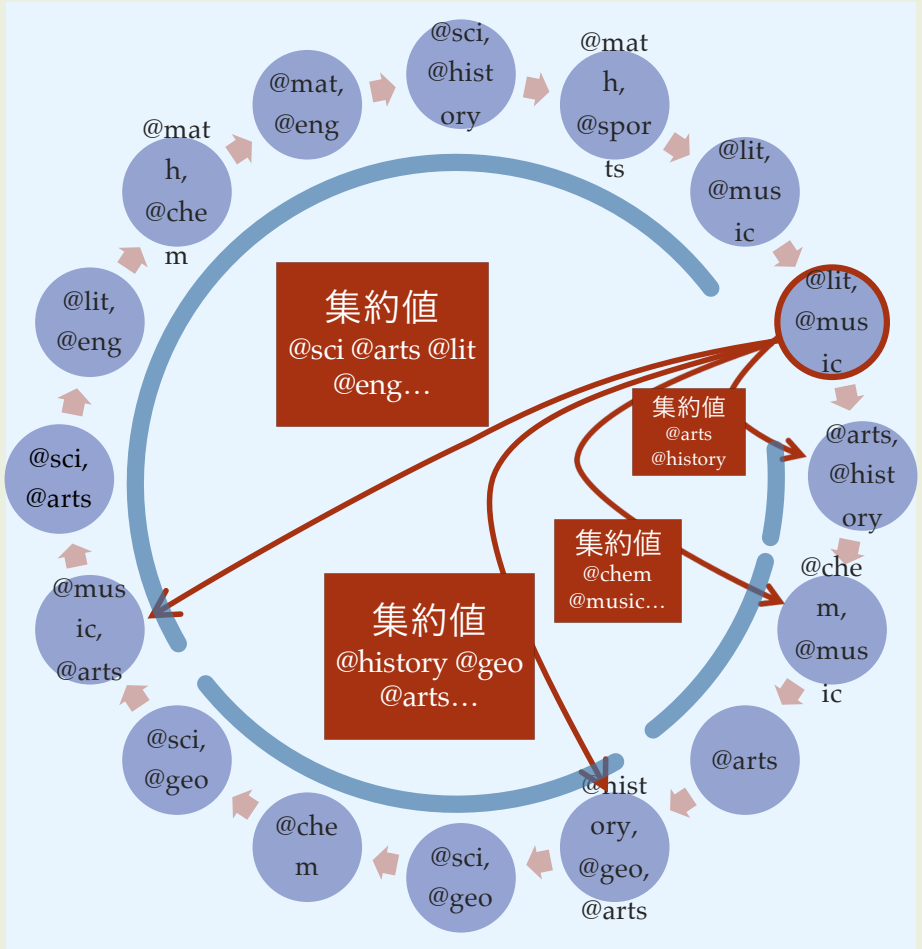


属性値の集約 (集約値) ①

- 区間内のすべてのノードの属性値を持つのは困難

➡ 区間内のノードの属性値をまとめて集約値に

- 集約値：区間中に対象ノードが含まれるかを判定できる値



属性値の集約 (集約値) ②

- 集約値はユーザ定義関数reduceで計算

  2つの属性値(or集約値)を1つの集約値にまとめる関数

  使用するmatchの条件に合うように定義

val = reduce(val1, val2)

reduceがみたすべき条件:

  val1 or val2 がmatch条件を満たす→ match(val) = true

- 例

reduce([@music, @sci], [@lit, @math])

  = [@music, @sci, @lit, @math]

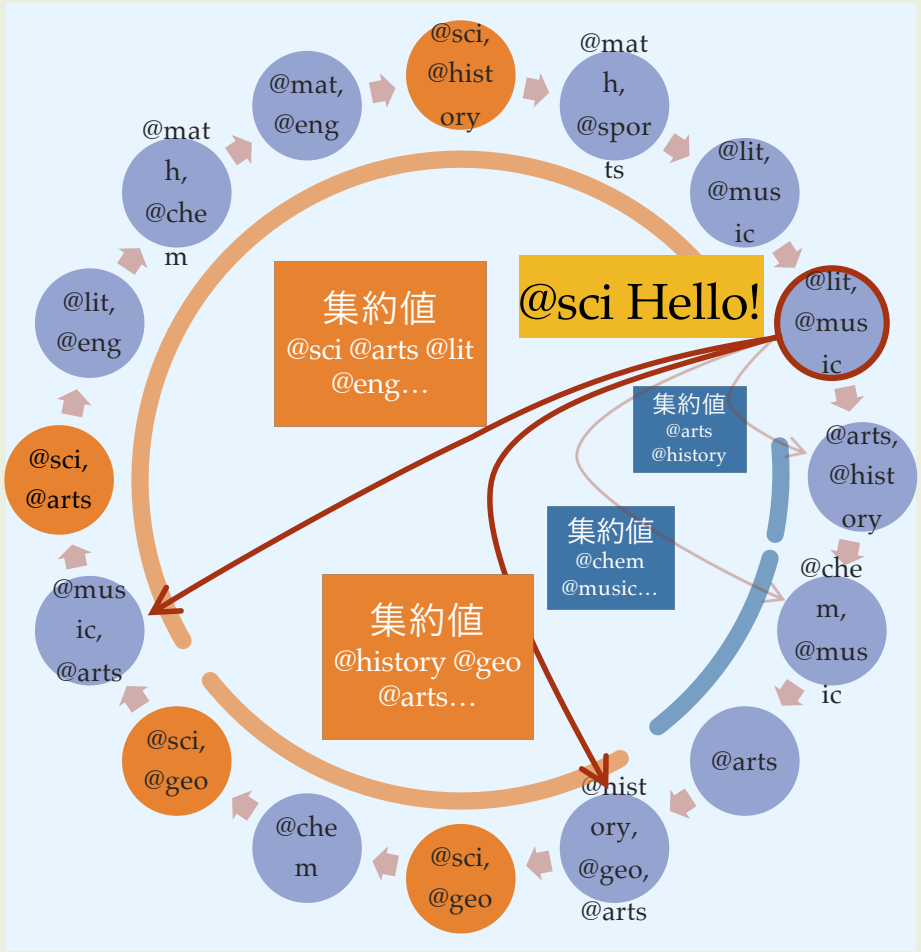
集約値の持ち方

経路表に集約値を保持

➡match(集約値)=trueの区間だけ配送

経路表

レベル	ポインタ	キー	集約値
-1	自ノード	自分のキー 29	自分の属性値 @lit @music
0	+1離れたノード	36	@arts @history
1	+2離れたノード	40	@chem @music..
2	+4離れたノード	47	@history @geo @arts…
3	+8離れたノード	67	@sci @arts @lit @eng…

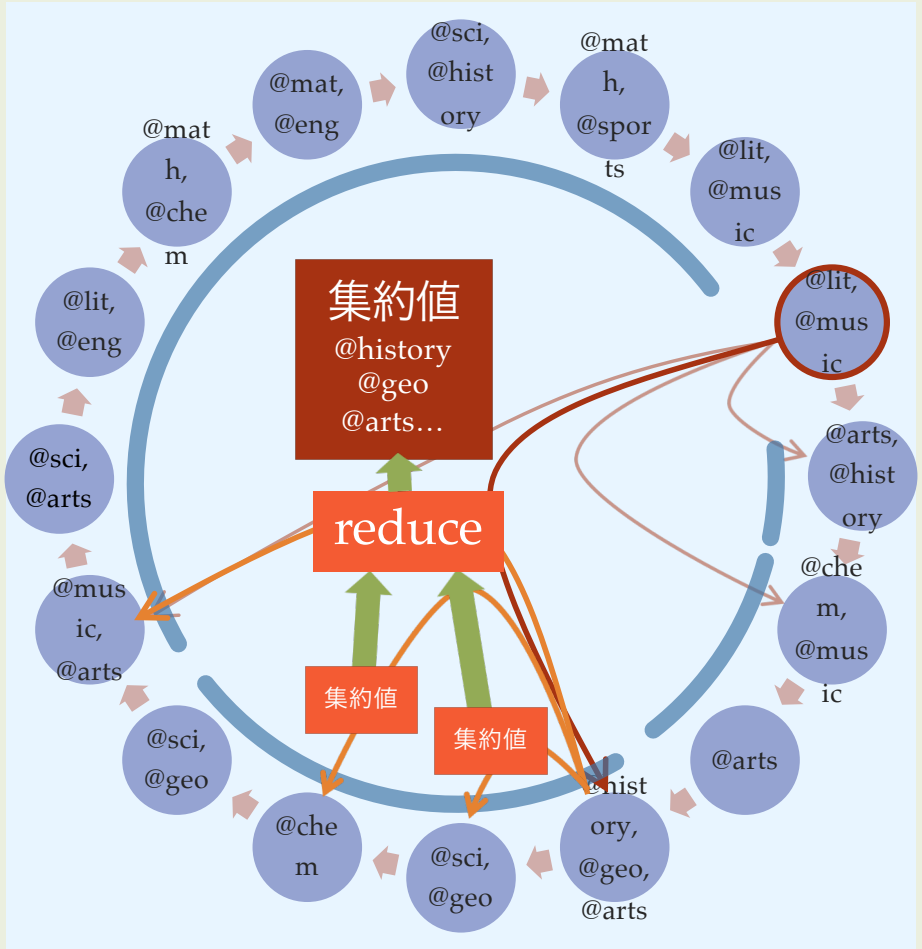


拡張部分

集約値の収集①

- レベルiの集約値＝
  レベルiのリモートノードの
  レベル[-1]～[i-1]の集約値の
  集約値
- すべてのレベルで更新

レベル	ポインタ	キー	集約値
-1		自分のキー 29	自分の属性値 @lit @music
0	+1離れたノード	36	@arts @history
1	+2離れたノード	40	@chem @music..
2	+4離れたノード	47	@history @geo @arts…
3	+8離れたノード	67	@sci @arts @lit @eng…



集約値の収集②

- 各ノードは，定期的に経路表のレベル0から最大レベルまでの集約値を収集

  Chord#の経路表更新のためのメッセージを拡張

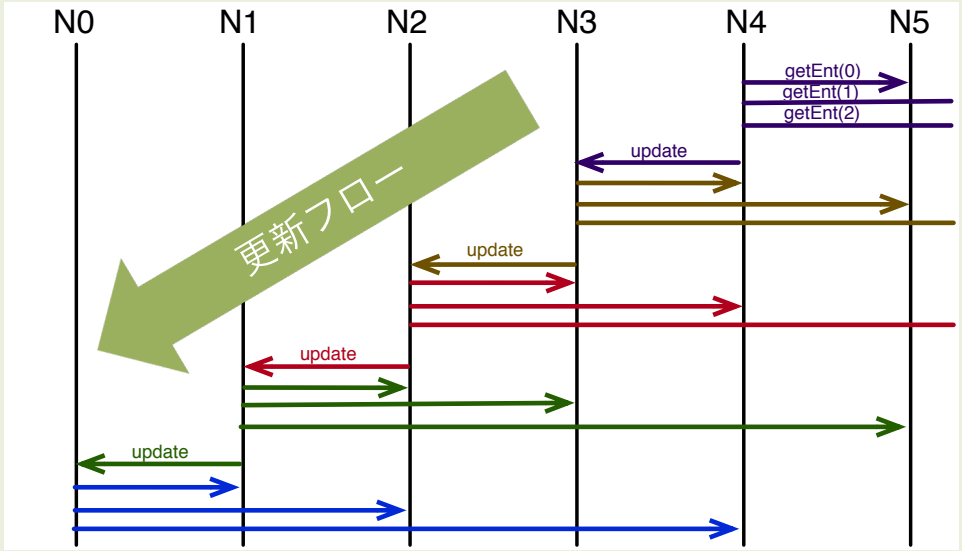
- 各ノードが任意のタイミングで収集しても動くが，より効率が良い収集方法を考案

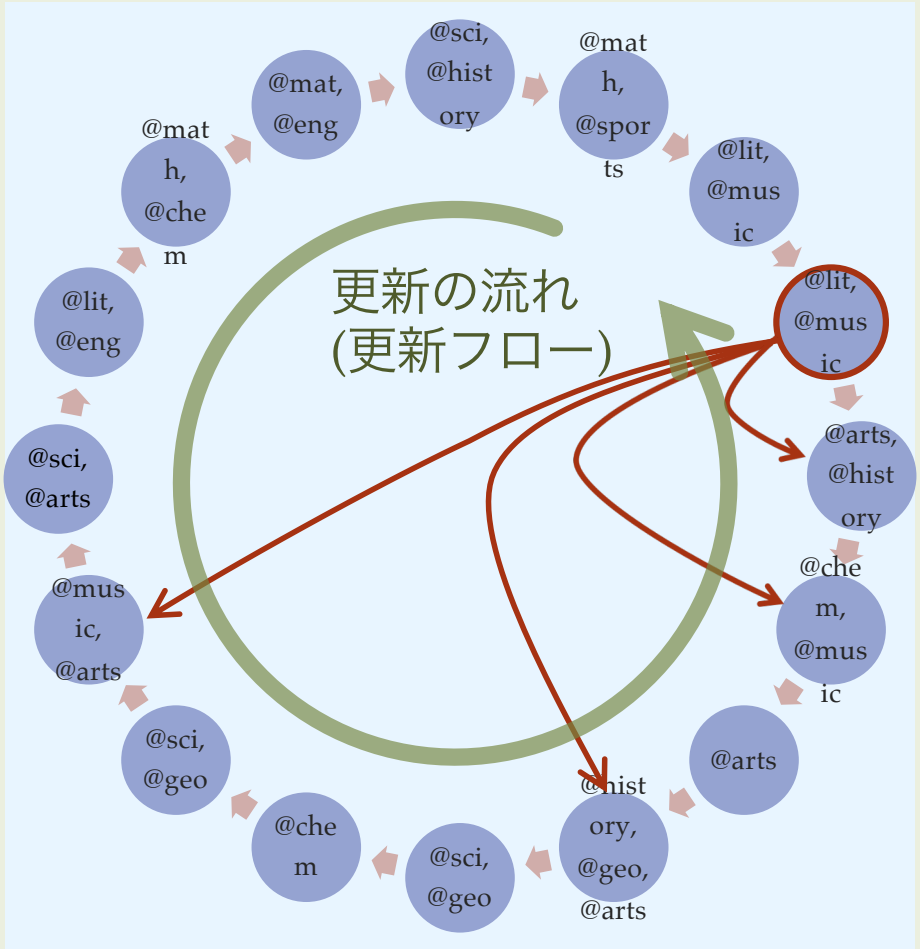
効率の良い集約値収集手法

効率の良い集約値収集

- 右ノードから左ノード(反時計回り)に収集

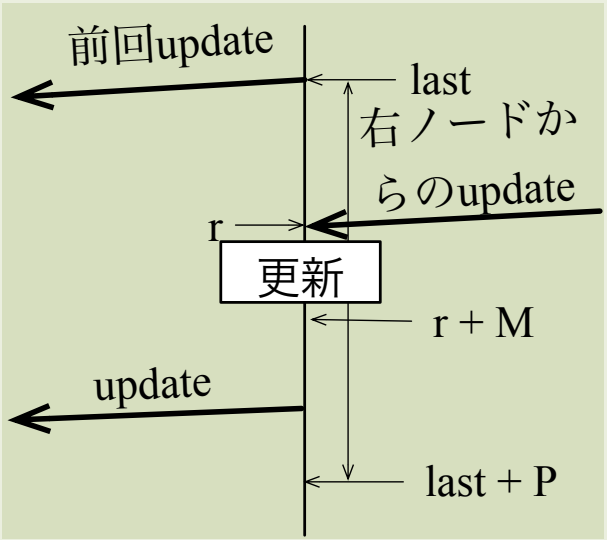
  右ノードからupdateメッセージを受信したら，すべてのレベルの集約値を収集し，左ノードにupdateを送信





効率の良い集約値収集 | タイミング

- 各ノードの更新間隔が一定(P)になるように自律的にupdate送信タイミングを調整
- 相反する目標を同時に満足する



① 前回update送信時刻+Pで送信したい

② updateを受信したらなるべく早く左ノードに送信したい

最小遅延時間

左ノードに送信する時刻 $= \alpha(\text{last} + P) + (1 - \alpha)(r + M)$
(where $0 \leqq \alpha \leqq 1$)

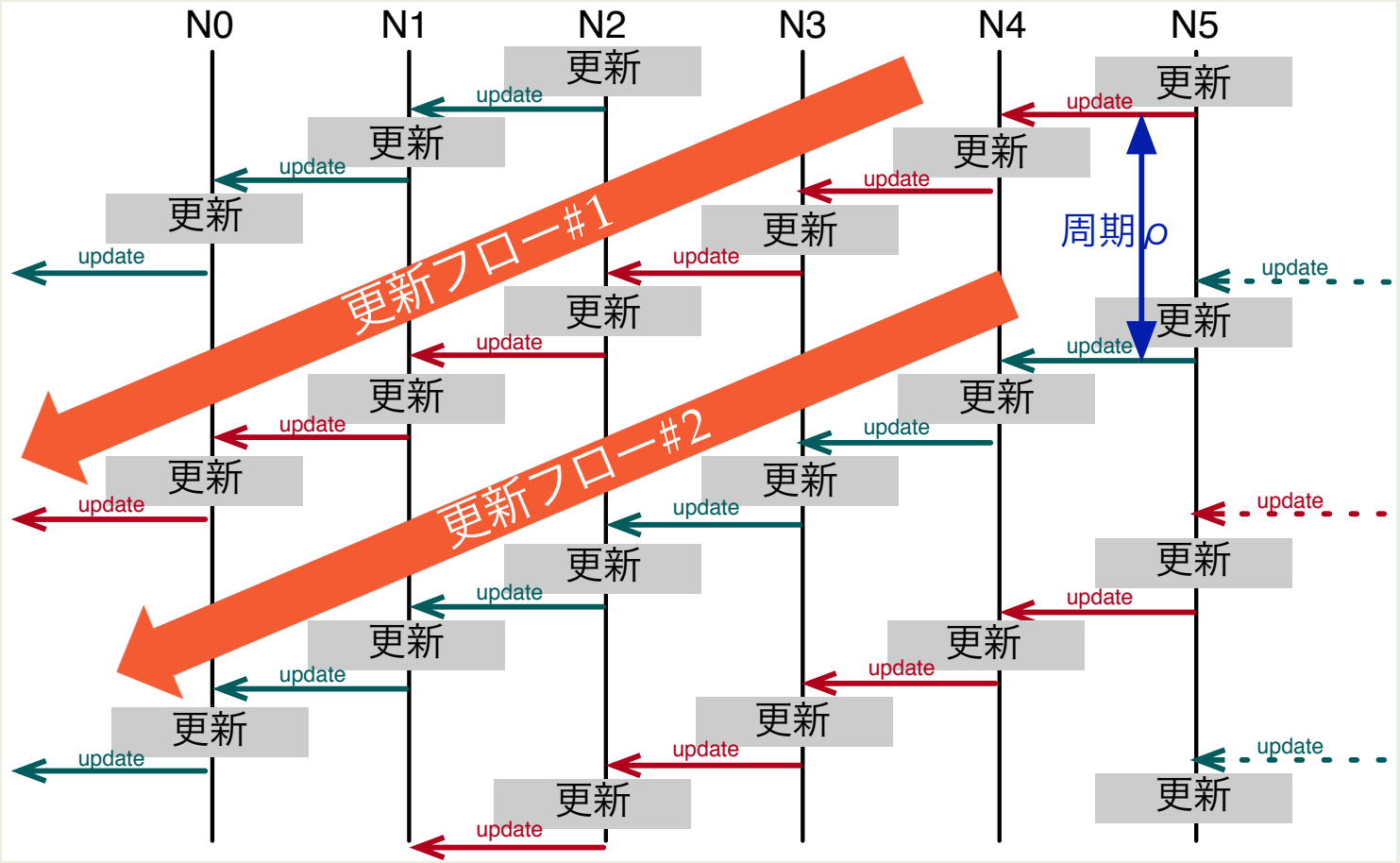
前回update送信時刻

今回受信時刻

徐々に理想に近づける

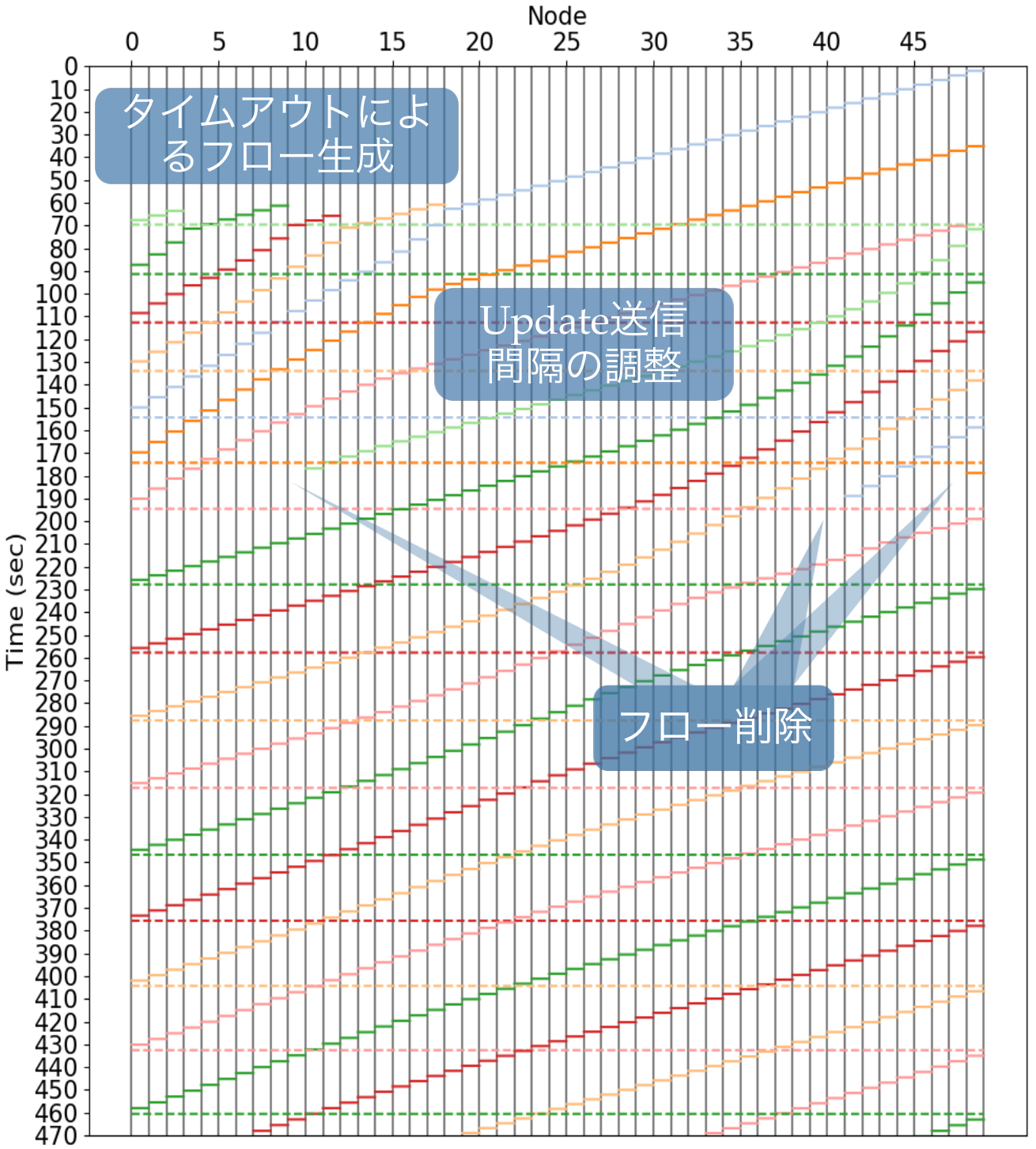
効率の良い集約値収集 | フロー数の調整

- 周期を守れるように自律的にフローを増減



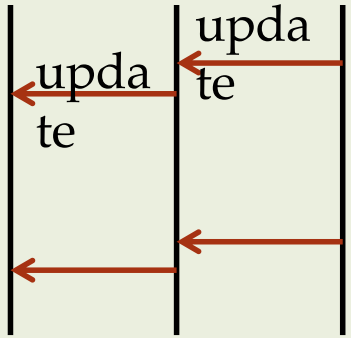
効率の良い…シミュレーション

- 50ノード一斉挿入
- 目標更新間隔30秒
- フロー数4, 更新間隔28.9秒で収束

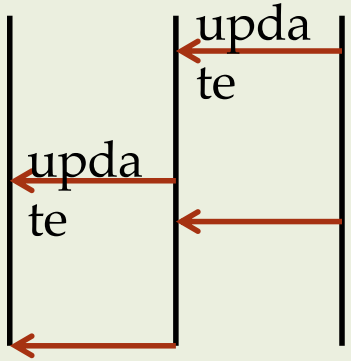


効率の良い集約値収集 | 解析①

- フロー数は多い方が良いのか少ない方が良いのか?



フロー数小



フロー数多

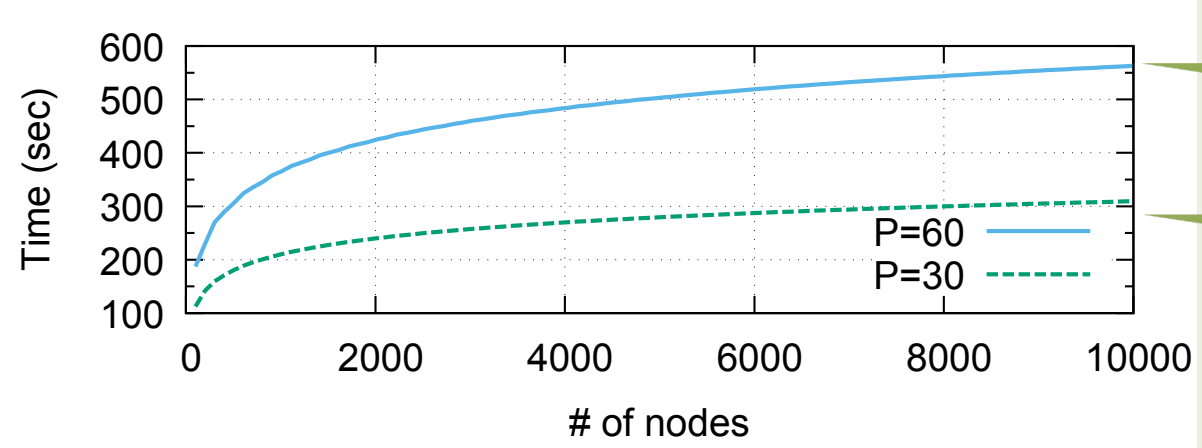
どちらも同じ
更新間隔

わかったこと

- フロー数は少ない方が良い ($F = \lceil Mn/P \rceil$)
- すべての集約値の収集するために…
  - 必要な時間は $O(\log n)$
  - 必要なメッセージ数は $O\left((\log n)^2\right)$

効率の良い集約値収集 | 解析②

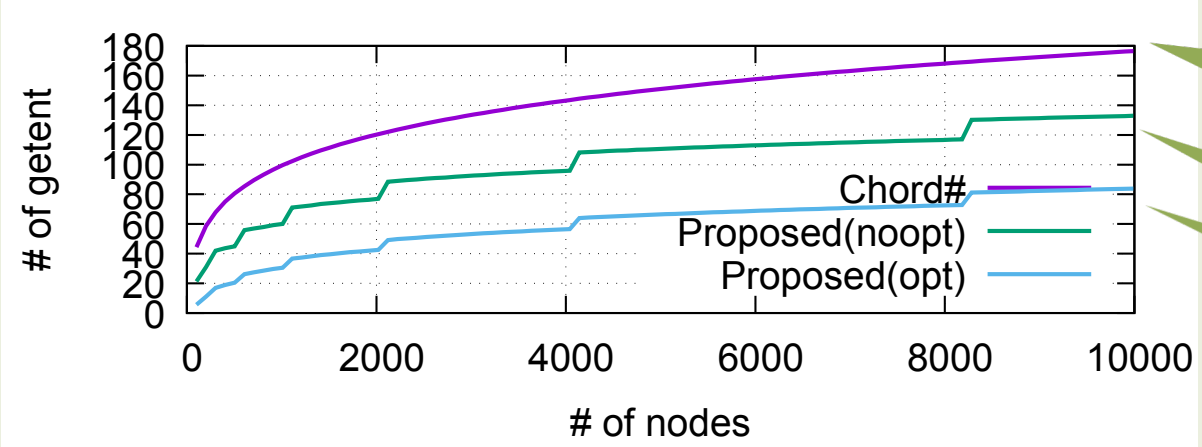
- すべてのノードの属性値の収集にかかる時間



周期 = 60s

周期 = 30s

- すべてのノードの属性値の収集にかかるコスト



勝手なタイミング
(Chord# と同じ)

提案手法

提案手法+最適化

応用例

結局，何ができるのか

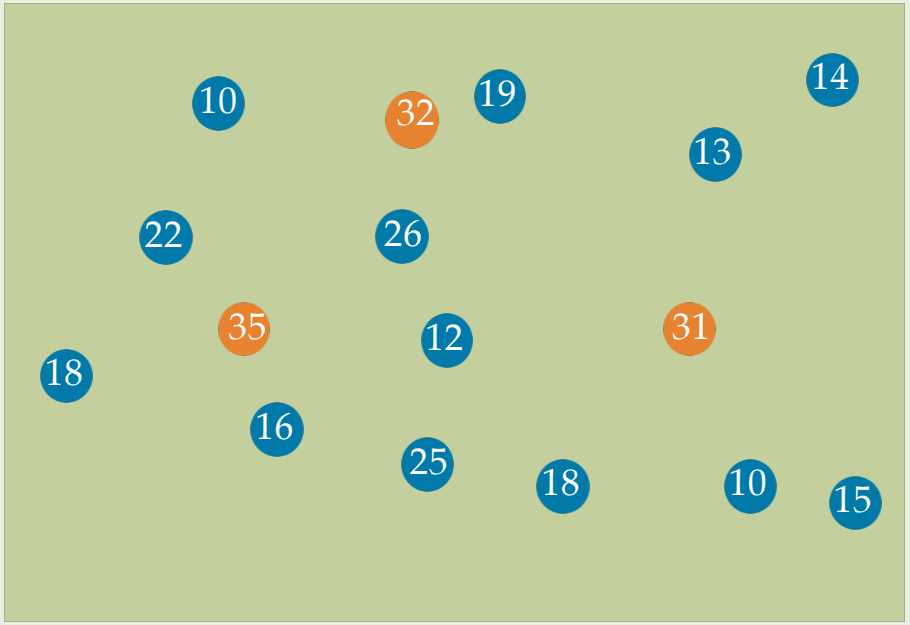
応用例

- コンパクトで偽陽性が低い集約値があれば理想的

  偽陽性：集約値では区間内に対象ノードが存在する(match()=true)が，実際には存在しない場合

属性値がC以上のノードへマルチキャスト

- 集約値 = スカラー値
- match(value) ≡ (value ≧ C)
- reduce(v1, v2) ≡ max(v1, v2)



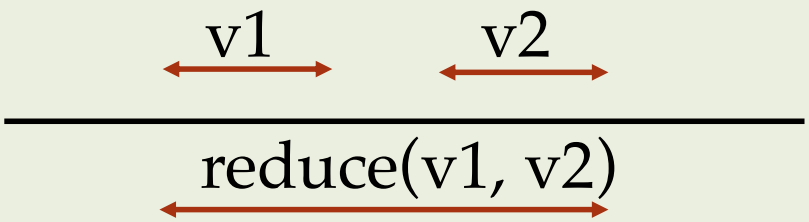
(例1) 30°C以上のノードにマルチキャスト

ノードの値が範囲内のノードにマルチキャスト

- 各ノードu が値 u.c を持つとき,
  値が範囲 r = [min, max) 内のノードに
  マルチキャスト

  - 集約値 = 1次元範囲
  - ノードの属性値 = [u.c, u.c]
  - match(value) ≡ value と r に重なりがあればtrue
  - reduce(v1, v2) ≡ 範囲 v1 と範囲 v2 を包含する範囲

- 多次元でも使える



特定のグループへのマルチキャスト

- 各ノードは1つ以上のグループに所属するとき，グループにマルチキャスト

  トピックベースPub/Sub

  - 集約値 = ビットマップ
  - ノードの属性値 = 所属グループを表すビットマップ
  - match(value) = (value[宛先グループのビット] == 1)
  - reduce(v1, v2) = v1 | v2　(ビット単位の論理和)

- 同時に複数のグループに送信も可能

特定のキーワードを持つノードへのマルチキャスト 27

- 各ノードが1つ以上のキーワードを持つ
- 属性値にBloom Filterを使う

  集合をビット列で表す確率的データ構造

  - 集約値 = Bloom Filter
  - ノードの属性値 = 保持するキーワードのBloom Filter
  - match(value) ≡ (BF(value)が指定したキーワードを含む)
  - reduce(v1, v2) ≡ v1 | v2 (ビット単位論理和)
- コンテンツベースPub/Subや全文検索などに応用可能

組み合わせ検索(AND検索・OR検索)

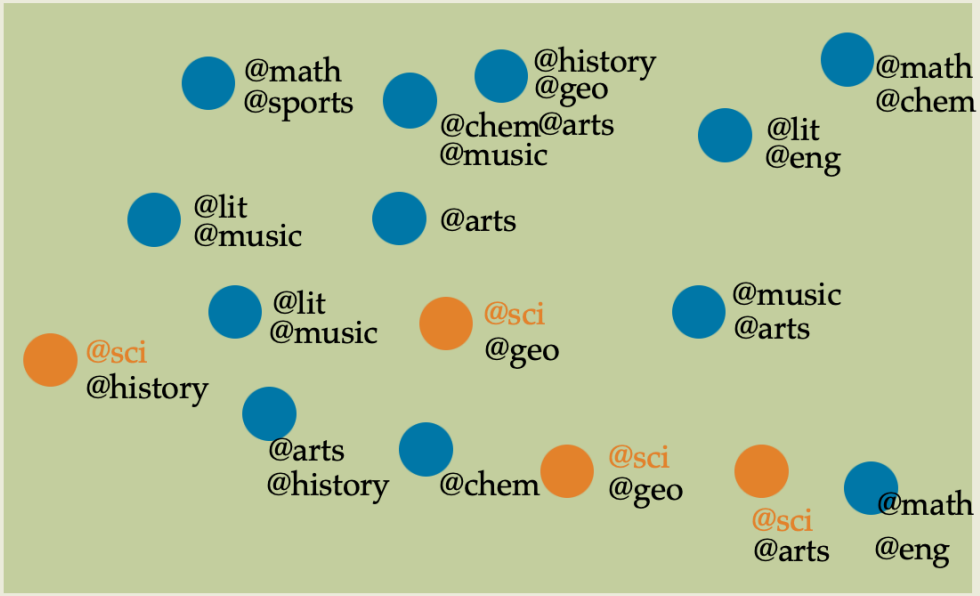
- ノードが複数の属性値を持つとき,
  組み合わせ検索(AND, ORなど)が可能
  (例: 温度と湿度, 位置(座標)と雨量など)
- reduceは属性値の種類ごとに定義
- あとはmatchの定義次第

- match(気温, 湿度) = 気温 > 30 AND 湿度 > 50

従来手法との比較

従来手法との比較①

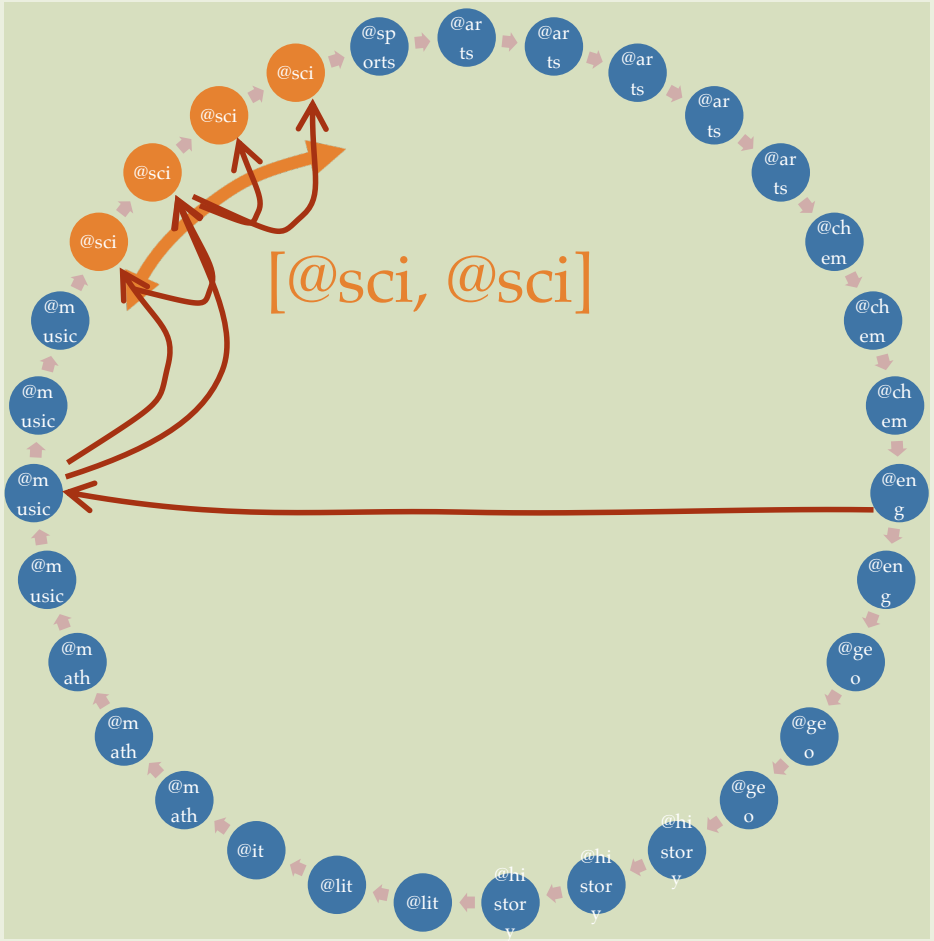
従来手法	• 属性値をキーにマップ • 適当なキー範囲にマルチキャスト



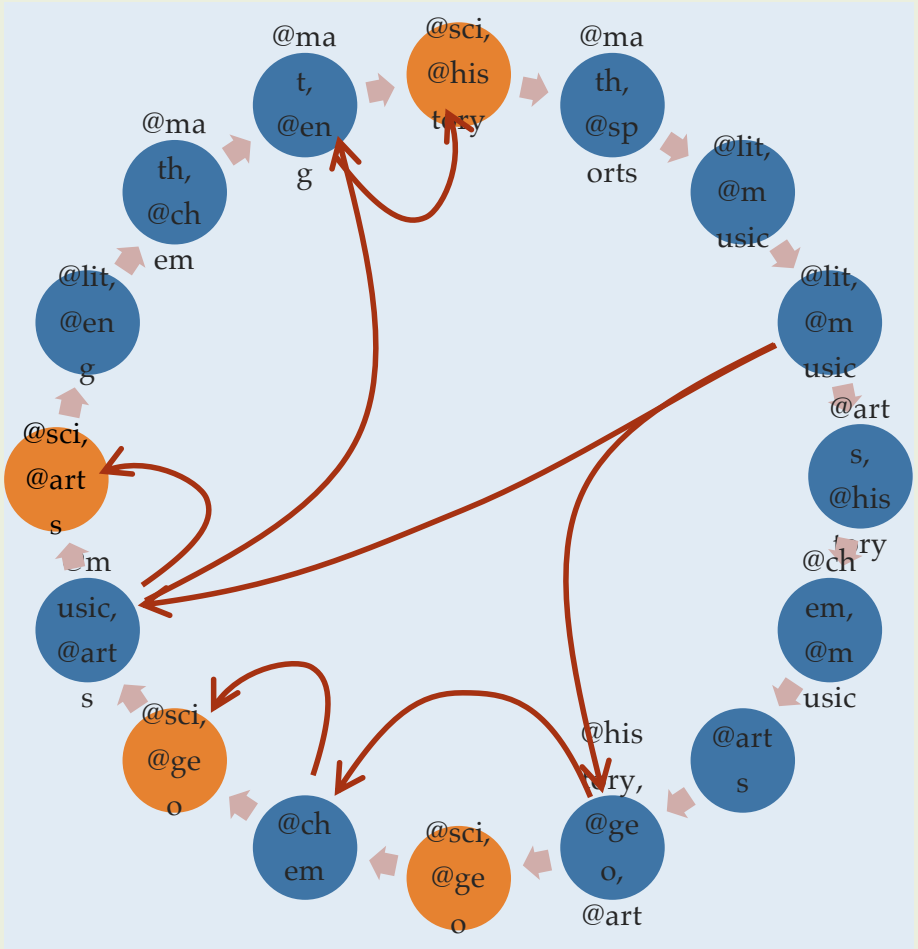
(例2) @sciの購読ノードにマルチキャスト
(Pub/Sub)

従来手法との比較②

従来手法
key=トピック名
(31ノード)



提案手法
key=なんでも (地理的位置など)
(16ノード)



従来手法との比較③

	従来手法	提案手法
柔軟性	× 属性値をキーにマップできない場合がある 属性値と別にキーが使えない AND検索は難しい	○ 任意の条件が可能 キーで絞り込みが可能 AND検索可能
ノードの挿入削除	× 属性値の変化毎	○ 1回
挿入するノード数	× 場合によっては複数	○ 1つ
マルチキャストの効率	○ 条件に合致するノードにしか送信しない	△ 偽陽性次第
即時性	○	× 集約値の伝播に時間がかかる
維持コスト	経路表を維持	定期的に集約値の収集が必要
最大ホップ数	O(log ノード数)	O(log ノード数)

まとめ

- 様々な条件付きマルチキャストが可能な方式を提案
  + 集約値を効率よく収集する手法
  適切な集約値が求められ，即時性が重要でない場合に適用可能
- 今後の課題
  即時性の改善(ノードの挿入や属性値の変化をアクティブに伝播)
  各種応用の評価
  処理を定期的かつ順番に行うアルゴリズムの使い道?