

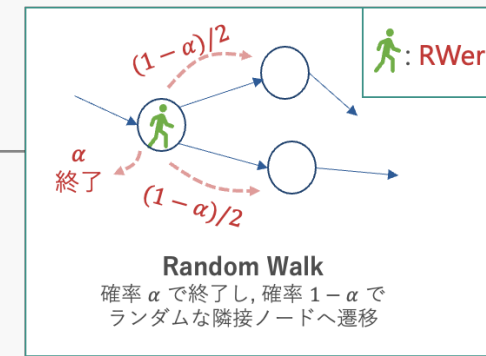
Random Walker 単位の非同期処理における Random Early Detection を用いた 自律的輻輳制御機構

滝沢 駿

慶應義塾大学

- **グラフ上での Random Walk (RW) の利便性**

- e.g., 推薦システム, コミュニティ検出, 類似性推定



- **既存手法：グラフの均等分割 & 同期時にデータ (RWER) をまとめて送信**

- 動的グラフを想定した場合, グラフ分割は不均等になっていく (負荷分散 ×)
- 複数ユーザによる多様なアプリケーション実行に不向き (同期タイミング ×)

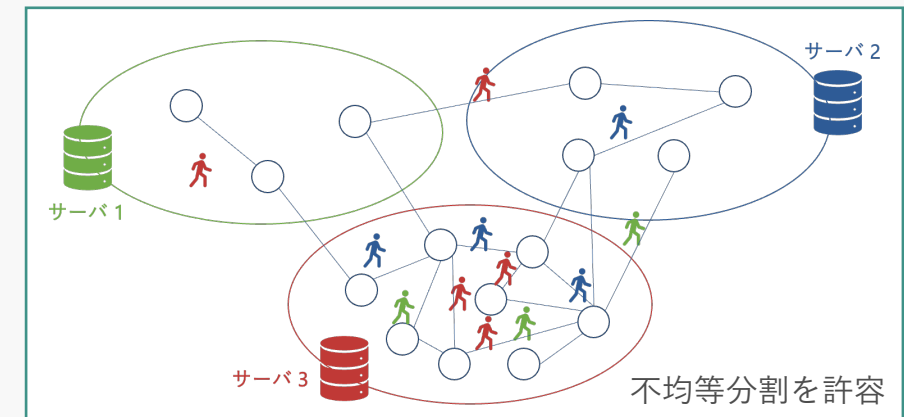
- **本研究では RWER 単位での非同期処理を採用**

- 同期待ちの解消により同期処理の弱点を克服

But...

- ✓ 一つのサーバに RWER が集中
- ✓ RWER キューのキューイング遅延が増加

RWER キュー：サーバが保持する RWER の待ち行列

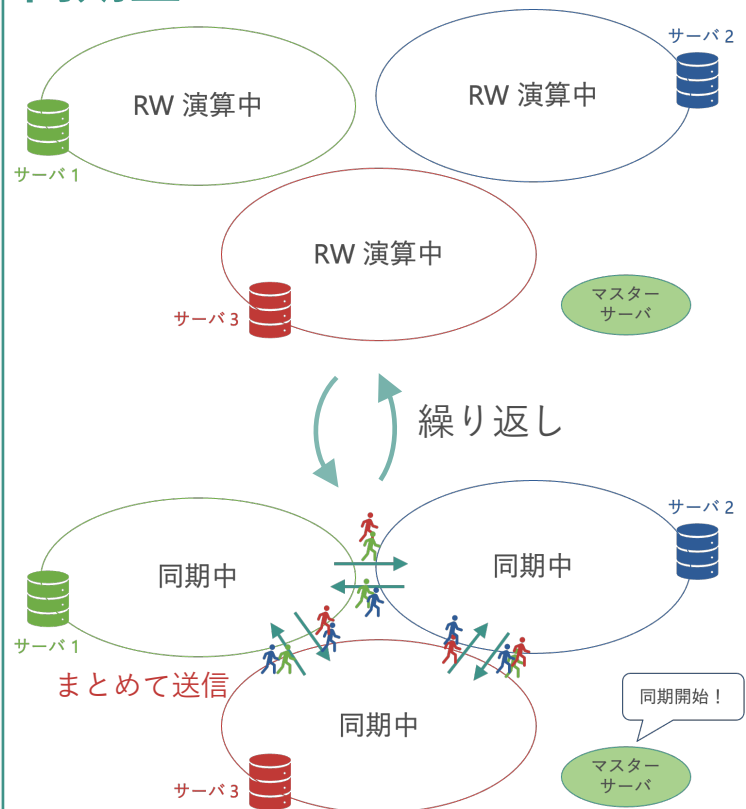


- **自律的 RWER 輻輳制御機構：各サーバが RWER の流量を自律的に調整**

- Random Early Detection^[1] を利用し, RWER キューを管理 → 輻輳の初期検知が可能に
- RWER ロスを検知し, RWER 生成スピードを落とす

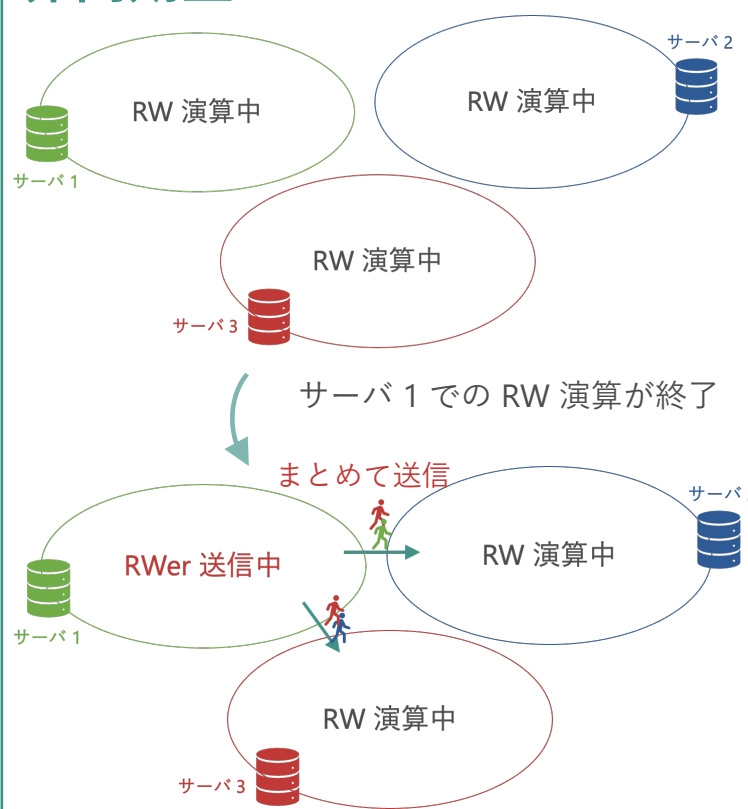
[1] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw., 1(4):397–413, aug 1993.

同期型[2]



- グラフの均等分割が前提
- マスターサーバが必要
- 早く演算が終わったサーバは同期まで遅いサーバを待つ必要あり

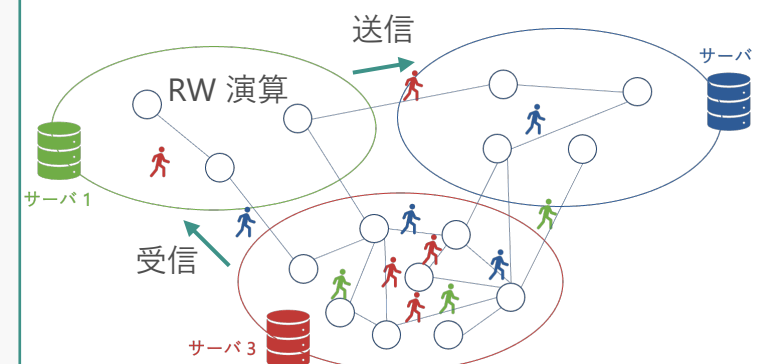
非同期型[3]



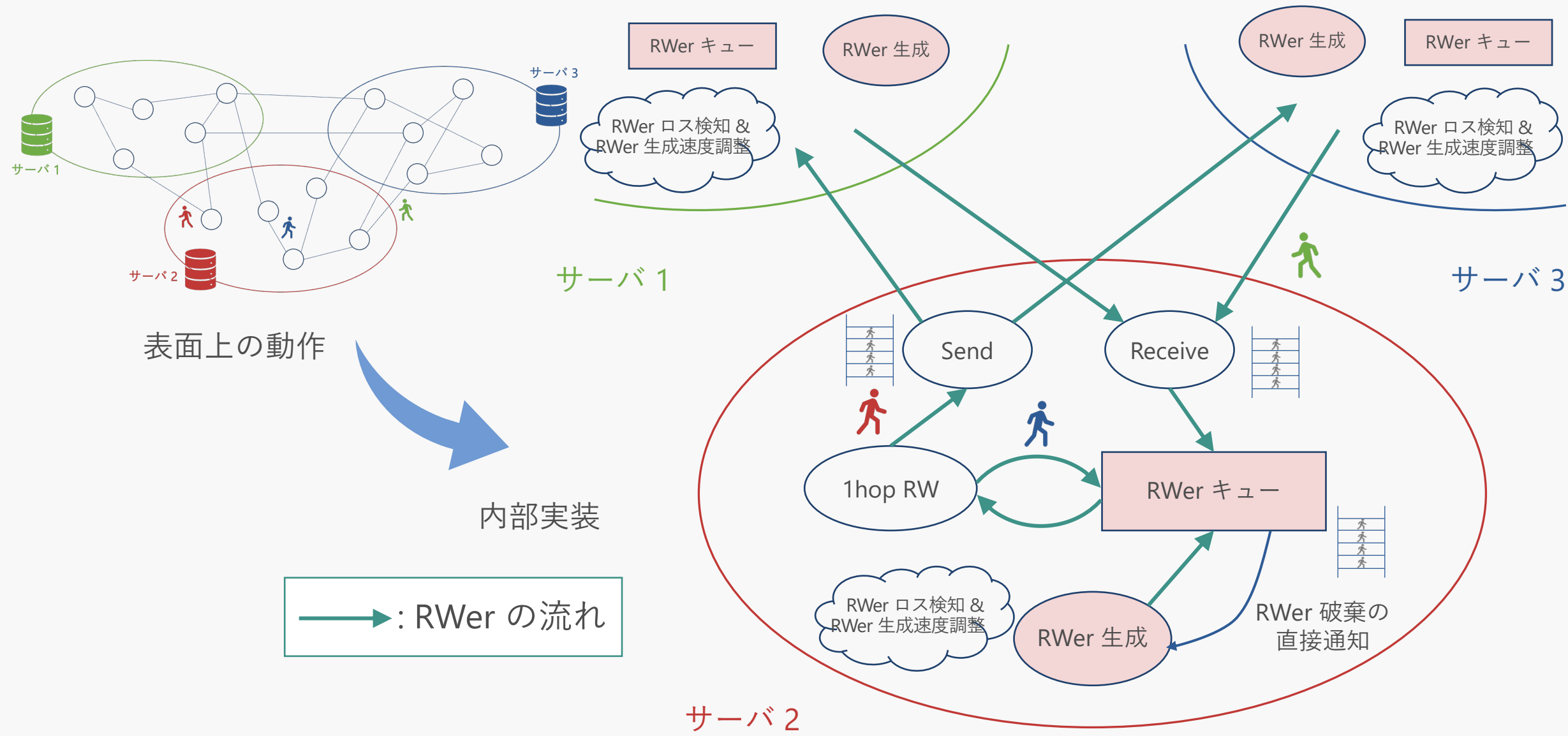
- グラフの均等分割が前提
- 早く演算が終わったサーバは他のサーバを待つことなく、次のステップへ
 - ✓ 同期型の待ち時間の発生を防止
- 依然としてサーバ内での待ちは発生

RWER 単位での非同期型

それぞれの RWER について、**独立して**演算, 送受信を行う

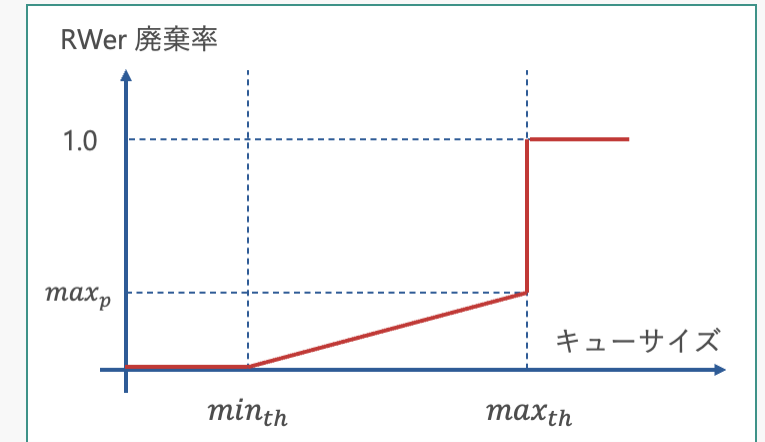


- グラフの不均等分割を許容
- サーバ内での待ちが発生しない
- 1 RWER = 1 パケットで UDP 通信
 - ✓ RWER の独立性を活かす
 - ✓ RW ではサーバを跨ぐたびに通信が発生 → TCP では無駄が多い
 - ✓ 1 RWER のデータサイズは UDP のペイロードに十分収まる
- **RWER が一箇所に集中する**場合がある
 - 本研究が対処する課題



キュー長に応じて RWer の破棄確率を決定

- Random Early Detection (RED)^[1] を参考に実装
 - ルータなどで採用されているキュー管理アルゴリズム
- テールドロップに比べ公平な制御
 - テールドロップ：キューが満杯のときに到着した RWer のみを破棄
 - RQ では輻輳の原因となる送信者の RWer 破棄率が自ずと上昇
- ランダムな RWer 破棄により **グローバル同期を防ぐ**
 - グローバル同期：複数のサーバが同時に RWer の生成スピードを下げることで、再び同時に生成スピードが上昇し、輻輳が発生
- キューが満杯になるより前に RWer を破棄し始め、RWer 生成速度調整を誘発させることにより、**輻輳を事前に防ぐ**



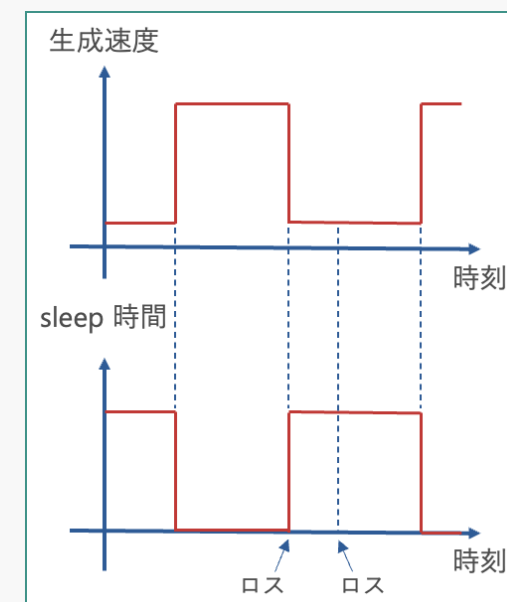
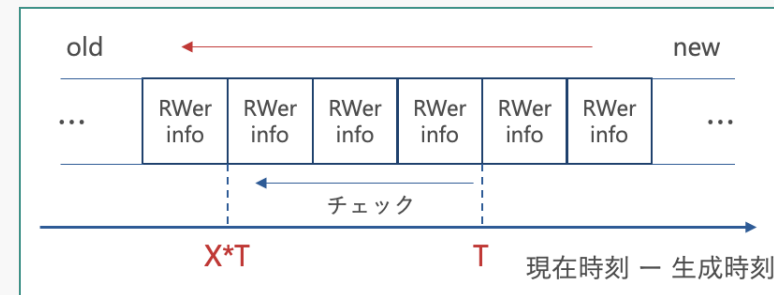
RWer ロスを検知し, RWer 生成速度を落とす

● タイムアウト検知と直接通知による検知

- 生成から経った時間が一定範囲内にある RWer のうち, まだ終了確認が取れていないものをロスしたものとみなす
- RQ で破棄した RWer の起点サーバが今いるサーバの場合, 起点ノードに直接通知し, ロスを検知させる
 - RTT の時間差なしでロスの通知が可能

● RWer 生成速度制御

- x 回以上連続で ロスを検知しなかったら $sleep = 0\text{ ns}$ それ以外は $sleep$ あり
 - 1~1000ns のスリープは制御できないので $sleep = 1\text{ ns}$ と設定している
- ロスを検知したらリセット

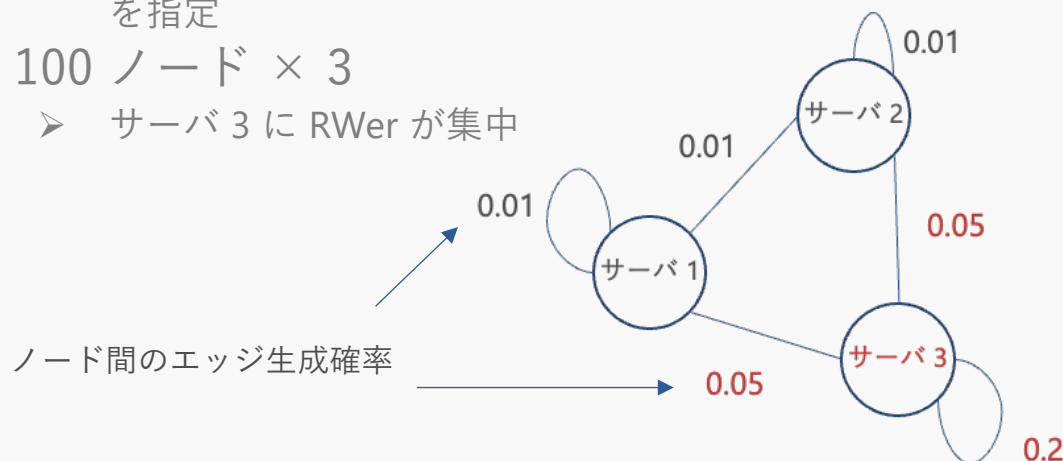


実装環境

- ① 3.5 GHz Intel Xeon 24 コア × 2 (サーバ 1, 2)
- ② 3.4 GHz Intel Core 8 コア × 1 (サーバ 3)
- C++ で実装

データセット

- SBM^[4] で生成
 - partition 数, ノード数,
partition 内のノード間のエッジ生成確率,
partition 間のノード間のエッジ生成確率
を指定
- 100 ノード × 3
 - サーバ 3 に RWer が集中

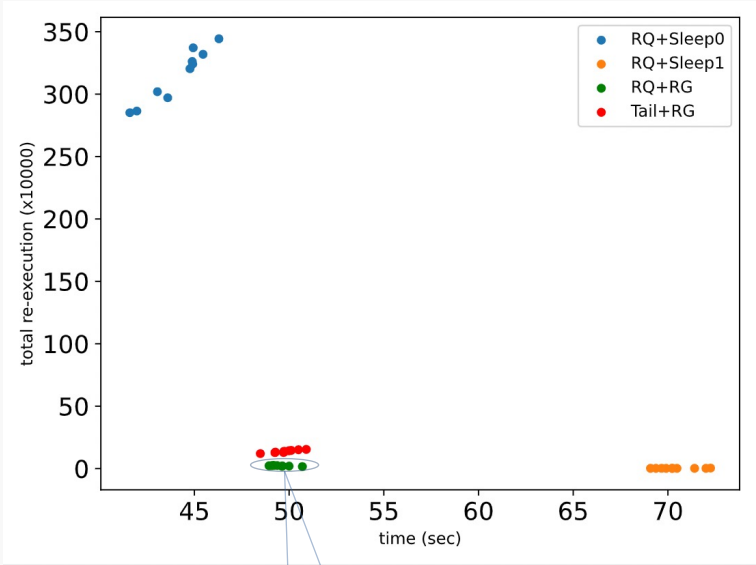


実験

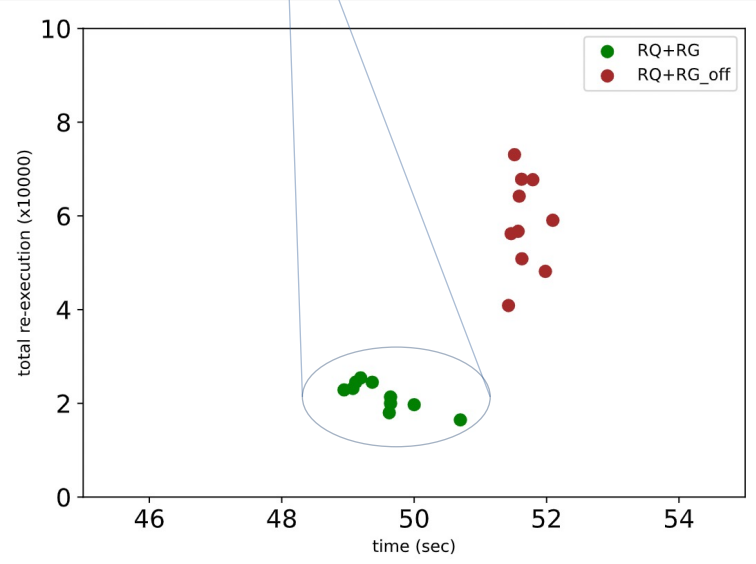
1. 全ノードから 10000 回分の RW が終了するまで実行を継続 (全サーバ同時開始)
 - (1 番遅いサーバでの) 実行時間, 総追加実行回数
 - ✓ RQ + sleep0 (スリープなし)
 - ✓ RQ + sleep1 (毎回 1ns のスリープ)
 - ✓ Tail (テールドロップ) + RG
 - ✓ RQ + RG
 - ✓ RQ + RG (起点サーバでの直接通知なし)
 2. 全ノードから 100000 回分の RWer を生成 (〃)
 - キューの状態とドロップした場所 (サーバ 3)
 - ✓ RQ + RG, Tail (テールドロップ) + RG
 3. グラフトポロジを変化させながら実行
 - 総追加実行回数
 - ✓ RQ + RG, Tail (テールドロップ) + RG
- ◆ パラメタ
- RQ : $\max_p = 0.1, \min_{th} = 5, \max_{th} = 100$
 - RG : $x = 3$
 - テールドロップ : $\max_{th} = 100$

RQ + RG : { 高スループット
低 RWer ロス } を達成

手法 (RQ + RG との比較)	実行時間	総追加実行数
RQ + RG	49 秒 (基準値)	20000 回 (基準値)
RQ + sleep0 (と比べて)	+ 5 秒	- 3,130,000 回
RQ + sleep1 (〃)	- 20 秒	+ 20,000 回
テールドロップ + RG (〃)	- 1 秒	- 120,000 回
RQ + RG (直接通知なし) (〃)	- 2 秒	- 40,000 回



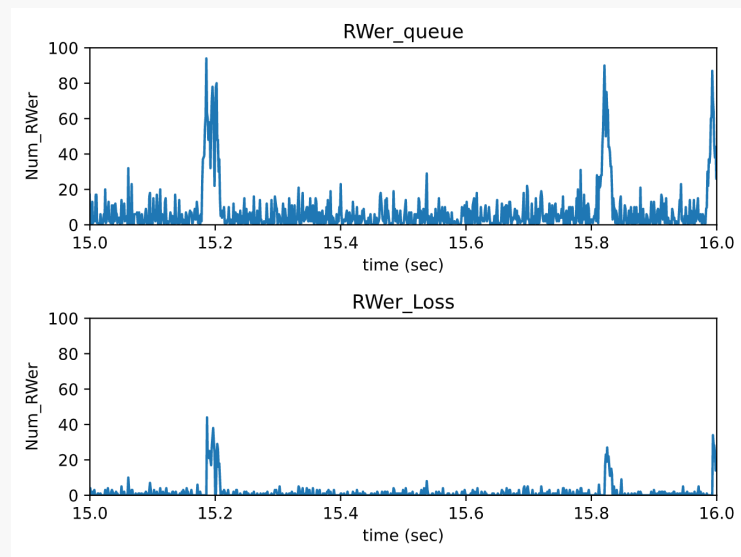
直接通知ありでの比較



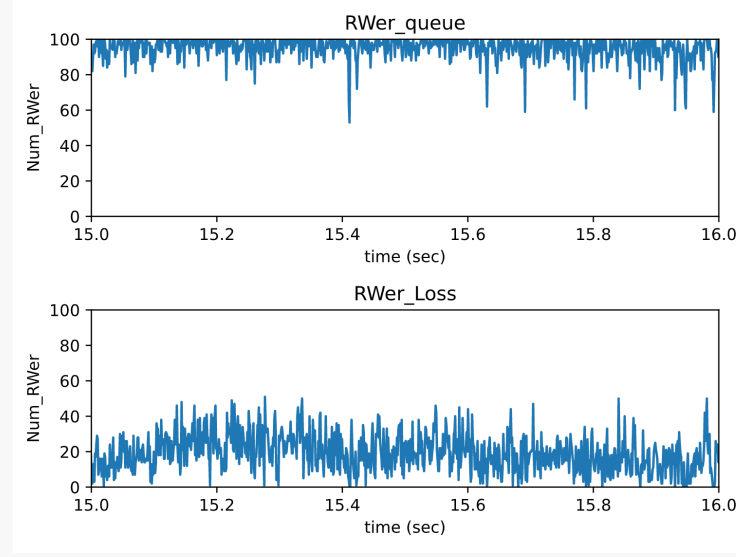
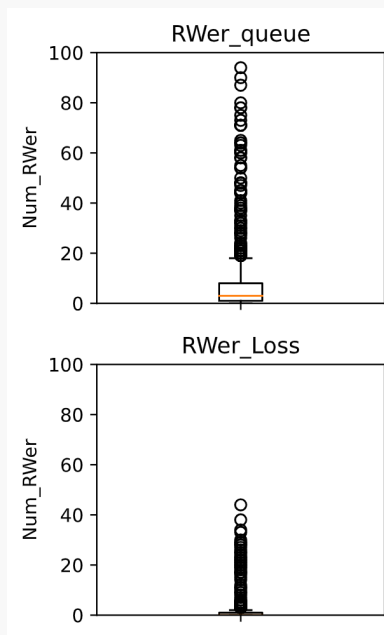
直接通知の有無の比較

RQ はテールドロップに比べ、
{ キューの平均長が小さい
RWerロスが少ない

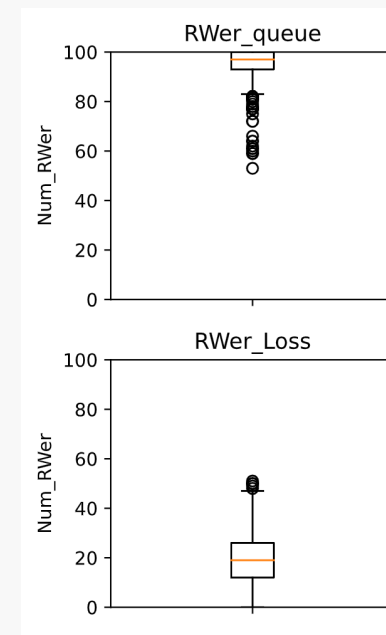
- RQ はキューが満杯になる前から RWer を破棄し始めるので
輻輳を未然に検知し、防ぐことができる



RQ + RG



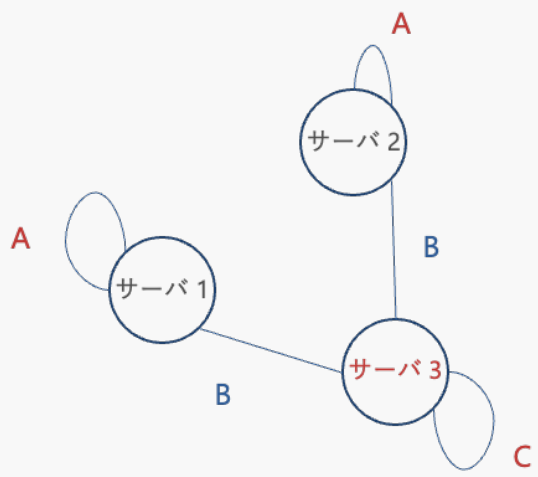
テールドロップ + RG



それぞれのサーバが 1000 ノードを保持

RQ はテールドロップよりも負荷耐性が高い

- RQ はテールドロップに比べグラフポロジに対する柔軟性が高い
 - B に対し C が大きくなるとロスが減るのは、サーバ 3 に存在するサーバ 3 で生成された RWer が増えることにより、直接通知による RWer ロス検知の効果が上昇するから



RQ + RG 「追加実行回数 (× 10000)」 テールドロップ + RG

<div>A : B \ B : C</div>	1 : 2	1 : 5	1 : 10
1 : 2	2.0	3.2	2.3
1 : 5	8.9	7.1	3.3
1 : 10	14.6	6.2	3.7

<div>A : B \ B : C</div>	1 : 2	1 : 5	1 : 10
1 : 2	13.6	17.6	17.3
1 : 5	21.8	22.2	22.5
1 : 10	25.6	24.1	26.1

自律的 RWer 輻輳制御機構：RWer 単位の非同期処理における輻輳回避システム

- **RQ**：キューが満杯になる前から RWer をキュー長に基づいて確率的に破棄
 - 輻輳を未然に検知
 - キューの安定化
- **RG**：RWer ロスを検知し、RWer 生成速度を調整
 - タイムアウトによる検知と直接通知による検知
 - RWer 生成間のスリープによって RWer 生成速度を調整
- **高スループットかつ低 RWer ロスを達成**
 - RWer 生成間のスリープをなくした場合に比べ、約 5 秒遅いが、総追加実行数が約 3,130,000 回少ない
 - RWer キュー管理をテールドロップにした場合と比べ、約 1 秒早く、総追加実行数が約 120,000 回少ない

- **実装の整理**
 - 先々のことを見据えた実装
 - RWer 等の抽象化
- **サーバ台数を増やして実験**
 - サーバ数が増えたときの挙動
- **RWer の送信に関する検討**
 - 現在は 1 パケット = 1 RWer
 - 1 パケットあたりの RWer 数を増やしてみる
- **RWer 生成戦略の検討**
 - 生成間のスリープではなく一度に生成する RWer 数を調整

